the world of

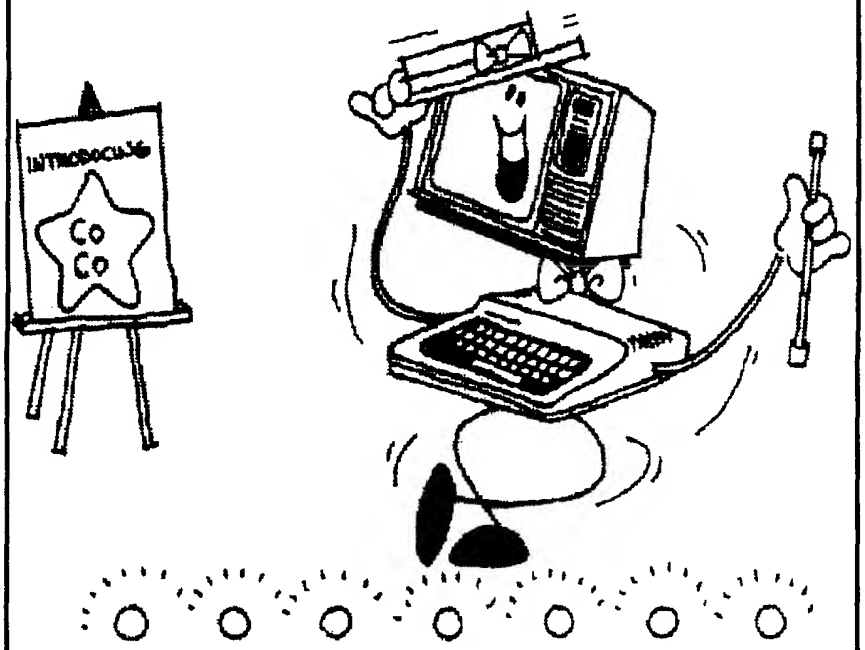
68' micros

Support for Motorola Processors

September 1994 *Vol. 2 Number 2* *$4.50 Canada, $4.00 US*

Happy Birthday CoCo...

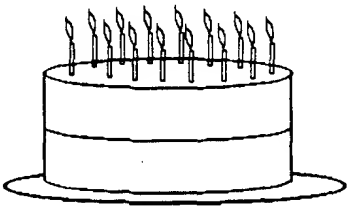
14 years old!



Also in this issue...

Use your Speech/Sound Cartridge with Level II
Using a hard drive with DECB
Generate true random numbers every time

CONTENTS



POSTMASTER:
If undeliverable return to:
FARNA *Systems* PB
Box 321
Warner Robins, GA 31099

Address Correction Requested

the world of 68' micros

Published by:
FARNA Systems
P.O. Box 321
Warner Robins, GA 31099-0321

Editor: F. G. Swygert

Subscriptions:
$23/year (8 issues) US; $30/year for Canada/Mexico ($12 US, $16 C/M for six months- four issues). Overseas $35/year ($18 for four issues) surface. Add $8/year, $4 six months for air mail.
microdisk: $40 per year, $21 six months, or $6 per issue. Overseas add $10/year, $5/six months, $1/single issue for air mail delivery. microdisk contains programs and source listings from each magazine; not stand-alone.

Advertising Rates:
$15 1/6 page, $20 1/4 page, $35 1/2 page, $60 full page, copy ready. Add $10 for special placement, $10 for typesetting ($5 1/4 or less). Dot matrix will be typeset if deemed unacceptable and submitter billed. 10% discount for four or more appearances.

The publisher welcomes any and all contributions. Submission constitutes warranty on part of the author that the work is original and not copywritten by another party. All opinions expressed herein are those of the individual writers, not necessarily the publisher or editor. FARNA Systems reserves the right to edit or reject any submitted material without explanation. Renumeration discussed on an individual basis.

Back issues are $4 per copy. Overseas add $1 each for airmail delivery.

Newsstand/bulk orders available. Dealers should contact the publisher for details.

Problems with delivery, change of address, subscriptions, or advertisers should be sent to the publisher with a short description.

The publisher is available for comment via **Internet** (dsrtfox@Delphi.com) or **Delphi** E-mail (DSRTFOX). The CoCo and OS-9 SIGs on Delphi are also frequented (The Delphi SIGs are sponsored by **Falsoft**).

The editor speaks...

F.G. Swygert

So we've gone through a full year together, at least the majority of the original 97 subscribers are still here! It has been a great year, with continued growth. I hope most of you do decide to continue your subscriptions in the future. Without you, there wouldn't be a magazine! And remember, this is YOUR magazine as much as mine because of your support!

If you don't think that your voice or seemingly simple complaint can't change anything, think again! Several of you complained about the lack of Disk BASIC support. I do hope to have more soon. One problem is a genuine lack of good DECB programs to print though. I don't want to write the entire magazine! I need your programs and ideas, comments, or whatever. So send in that little idea you have, that program you started, or whatever. If you have a good core program but are stuck on some problem, send it in anyway! I'll print what you have and accept suggested answers in future issues. This will go for ANY programming language, DECB or OS-9/OSK.

In one big way, the CoCo hobbyists are like the AMC auto hobbyists. The most vocal group are the performance people. They demand lots of coverage and are quick to send in articles and letters. They also make up the smaller part of the hobby (about 25%-30% of the AMC crowd). The people who support the hobby most through club memberships are the quiet bunch who like the more mundane, everyday transportation cars AMC was most known for. At one point, these prople split off from a large club because the performance crowd was complaining of not enough coverage in the newsletters. They were getting 50% or more of the coverage!

The point here is that the majority of CoCo users still use DECB to a large degree, they just don't write. They are content that their computers do what they were designed to do: provide good, everyday computing power at an affordable price. Some dabble in OS-9 or are beginning to learn it, others never will. But to maintain a good amount of coverage in the hobby publications and on BBSes, you MUST load that word processor or get out that pen and write! No matter how petty you think your discovery or thoughts on a matter are, you may find that others agree with you. If you want the coverage, you have to participate and be heard!

I know many of you noticed that the last issue was a little late, by about a week or week and a half. This was caused by two situations: 1) I took about two weeks (June 31 - July 19) for a summer vacation, and 2) my paternal grandfather died July 20. That took me about four days behind schedule, the vacation set me back about five. I'm sorry for the delays, but things DO happen! I'll try to keep things more on schedule in the future.

< 268'm >

NOTICE: ***The programming contest deadline has been extended until 01 January 1995! Take this time to work on those winter projects and get those entries in!!!***

~~PROGRAMMING CONTEST!!!~~

FARNA Systems has set aside some cash and prizes ($150 total!) for a programming contest!

ALL COMPUTER TYPES SUPPORTED BY 268'm ARE ELIGIBLE!

Send a disk with a running copy of the program as well as an ASCII listing of the source code or BASIC listing, running/installing instructions, and a descriptionto FARNA Systems PC, Box 321, WR, GA 31099 by January first, 1995. Programs may be of any type.

First place gets $50 cash.
Second place gets $25!

Letters to the Editor

Volume 2 number 1 (Aug '94) will be a "save for 10 years" issue. It is crammed with important and hard to get addresses. This effort will be on top in the front right drawer for quite some time.

James DeStefano
RD 1 Box 375
Wyoming, DE 19934

What can I say but "thanks", James! But save that "front right drawer" room... better issues are on the way!

Last year I decided t otavel down the path of a new progamming language -- "C". It wasn't until after I purchased the Tandy OS-9 C disks that I realized how difficult it was to find information on C. I would like to thank you for including articles in your magazine about the language. I know they have helped me a lot. I live in a rural area and it's hard to find a bookstore with information of that type. The big thing for me now is that I'm trying to write a pseudo-random number generator for my library. Any information on this would be greatly appreciated.

In your latest issue I see that the CoCo Family Recorder has been ported to OS-9. I have only one disk drive (FD-502). Would I be able to run this program?

I better stop now or I could fill a magazine myself. Heaven forbid I miss another issue of 68" micros!

Jay Duke
11642 S. Pines Trail
Roscommon, MI 48653

Jay, I'll have a random number program in DECB in this issue. Maybe that will give you some clues as to how to write one in C.

For your question about CCFR/OS-9, you're in luck! All you need to do is make sure your FD-502 is set as a double sided drive and the program will run fine. There is room on the disk for all data and program files, so you don't need to swap disks either. You can use CONFIG to set your drive as double sided... use the "40dd" drivers. You can atill read, write, and format 35 track SS disks.

For many reasons such as money, time, other interests, etc., I decided long ago not to purchase a modem and go on-line with any of the information services. Because of that I rely on the various publications for information, news of events, ads, new and used products, and all other news of the CoCo world.

Unfortunately, some of these publishers seem to be like ghosts, fading in and out of the woodwork I renewd subscriptions to two other publications in November and January and haven't received issues since the first of this year.

268'm has turned out to be the most informative and reliable of the CoCo publications. I want to express my appreciation to you for that. Keep up the good work! Please allow me to include the "Mid Iowa and Country CoCo Club" in the informative and reliable category as well.

Ray Watts
Box 574
Niantic, CT 06357

Ray, you will note that I didn't print the names of the two publications you mentioned. I hope you aren't offended, just consider it as a professional courtesy to them. If I get more complaints, I will print something later.

MI&CCC is indeed a relaible source of information, especially for DECB users. They have some OS-9 also, and are purely CoCo. If DECB is your main interest, you should definately give them a try. A double sided disk drive and 80 column monitor is required to view the "Upgrade" disk magazine.

Found on the Delphi OS-9 SIG....

>>Well it seems a lot of the OSK'ers have forgotten their roots and are now dogging us cocoists mercilessly<<

I don't think that's true. There are some situations, true, but the OSK users as a whole aren't doing such a thing. They ARE trying to persuade people who want more updated and supported and POWERFULL software to go to OSK for it rather than have it come to the Coco where it is unlikely to work well. In the process some may sound like they are dogging their roots so to speak, but the intent is to get people focused on where progress lies. If the Coco is still serving your needs, great, stick with it, but when people you want the OSK performance, you need to go OSK (am I the only one who thinks I am beating a dead horse with that statement?). I myself am still using the Coco and don't plan on ever getting rid of it, even after the day comes that I do go OSK. That day is inevitable as I am already getting anxious to jump into the OSK world where there is a possible future market, unlike the level II dieing market (it's only natural), so I can count on a greater number of quality applications being released (a trend which seems to be slowly but surely picking up), and so that I may someday be able to count on some financial return of any programs I may write and release in the future.

Keep in mind, I still have a great respect for the Coco, and always will. As has been stated a million times, it worked wonders compared to the other machines in it's day and still does impressive stuff, BUT the new machines were created with the hope that we would have something better and different to jump to besides the PC's when we had needs that the Coco wasn't capable of meeting. Now we have these machines, and yes a very modest base of apps, but a start nonetheless. I hear from a lot of CoCo diehards (no offense, I'm one of them), about how bad this comunity's going down the tubes because there are things we can only do effectively on the MM/1 and other machines but not the Coco. Isn't this the whole purpose of the new machines? Being a Coco diehard is great, but hopefully we can all learn to realize when we are ready for one of the new machines. This lack of realizing is what drives a lot of the OSK'ers nuts.

Chris Perrault
CPERRAULT on Delphi

Chris's letter gives us a lot to think about. That fact is what impressed me enough to print it here...

68' micros is top notch. I have published newsletters before, so I have an idea how much dedication and effort it must take you to publish your quality magazine, the design and content of which are super.

I really enjoyed your article on "OS-9/68000 on the Cheap" in the August issue. I am not a hacker, but I am trying to learn. My dad gave me an old Kaypro and I bought a Tandy 1000HX cheap from a friend. I'm not using these and was interested to read that old 8088 based machines make good platforms to build a OS-9/68K system from. I may decide to build an OSK machine later.

Ted Willi
Box 447
Athens, GA 30603

Ted, if the Kaypro is a portable, a CoCo 3 motherboard fits nicely (see Vol. 1 #1 & #2). The 1000HX would also make a good home for a CoCo3, but not so good for a 68000 board. I don't think the 68K boards would fit, though a KiX/20 might (should also fit the Kaypro case, but monitor would need changing). Tandy clones aren't good choices: they used odd sized motherboards, but some (original 1000) could be modified to mount a "standard" XT size motherboard and cards. The EX and HX won't hold any expansion cards, required for video by all the 68Ks.

< 268'm >

Letters are printed on a space available and popular subject matter basis. If you don't want your letter printed, or wish to withhold your address and/or name, please state so when writing. In some cases, letters are edited for space and/or clarity. If a personal reply is desired, please enclose an SASE.

Fourteen Years of Color Computing

F.G. Swygert

We celebrate a truly historic occasion... the introduction of the first CoCo!

On July 31st, 1980, Tandy publicly introduced its new computers for 1981: the TRS-80 Model III, TRS-80 Pocket Computer, and the TRS-80 Color Computer. It wasn't until September that the CoCo actually became available in Radio Shack stores... sort of like a 30-40 day pregnancy. You know it's coming, it just isn't quite there yet. So it is that I decided to "celebrate" the birthday in the September issue.

This first machine would be laughable by today's standards, but at the time was at least on a par with the competition (Apple II, Atari 400, and Commodore VIC-20). It sold for a reasonable $399.00 and came with 4K of dynamic RAM, 8K ROM with Microsoft Color BASIC 1.0, a 53 key calculator style keyboard (often referred to as a "chicklet" keyboard), built-in video modulator so a TV could be used easily, 16 line by 32 character display, an RS-232C serial port, 1500 baud cassette interface, two joystick ports, and an expansion/cartridge slot.

Unfortunately, Radio Shack was never the place to find good info on the CoCo. The CoCo was sold only to those with more of a gaming , educational, or "home use" interest, or simply refused to (or couldn't) spend more money on a TRS-80 (and later Tandy produced IBM clones). The TRS-80 and IBM clones, with their many business applications and widespread acceptance in the business community, were the serious (and more expensive... likewise more profictable for the company) choice. Since the company didn't really push the CoCo, it was up to users to discover its hidden potential. Once a few people discovered the hidden values of the CoCo, it almost sold itself.

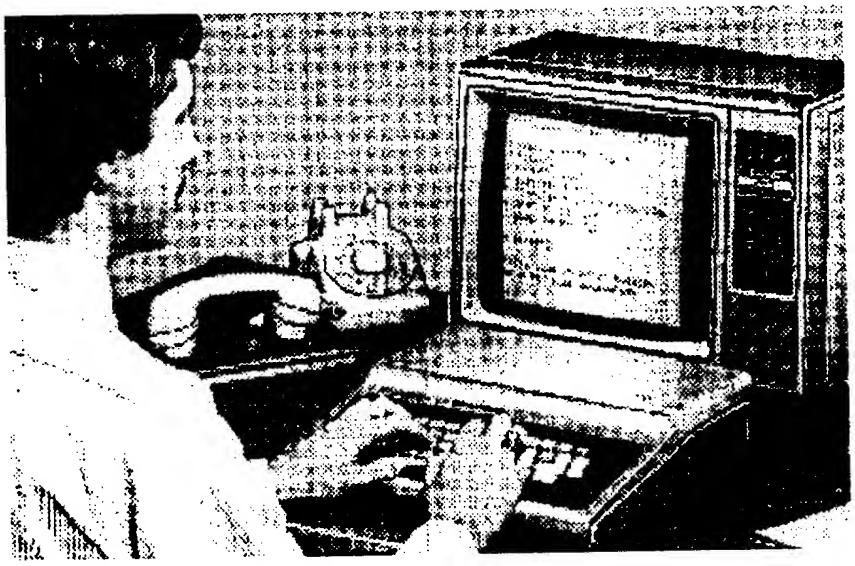
It was the independant, third party software and hardware developers (who were actually discouraged by Tandy until it was to late) that really made the CoCo. Small "cottage industry" companies sprang up to fill the software, hardware, and information gap left by RS: The MicroWorks (Andrew Phelps), Spectral Associates, Computerware, Colorware (Mickey Ferguson), Frank Hogg Labs (Frank Hogg), Tom Mix, Chromasette Magazine (Dave Lagerquist), The Rainbow Magazine (Lonnie Falk... at that time nothing more than a newsletter), Cer-Comp (Bill Vergona), Green Mountain Micro (Dennis Kitsz), Derringer Software, Spectrum Projects (Bob Rosen); and this is just a drop in the bucket of some of the most recognized names to us long-time users. Of these, only Cer-Comp, and Colorware (as Alpha Software) remain in the Color Computer software business. All started within the CoCo's first three years on the market.

The last Color Computers, CoCo 3s, were manufactured in 1990. Tandy continued selling built-up stock throughout 1991, discounting both hard and software drastically. Many CoCos, peripherals, and software was still available in RS stores nationwide as late 1992, when were-is-as-is sales and managers specials depeleted stocks.

Luckily, Tandy continues to sell nearly all software they produced through their Consumer Mail Center catalog. If you need a former RS title, contact your nearest dealer and ask to see the CMC catalog. The book can't leave the store, however. If you have an old catalog number, that will help. Most software comes with a xeroxed or laser printed manual from the Tandy archives. Prices are usually lower than the last published catalog prices.

The CoCo lives on for many of us. It will continue to do so, running both the built-in BASIC and Microware's OS-9, for many years to come. It was, despite Tandy's inability to recognize its true power and market potential, the most successful of the "home" computers.

< 268'm >



Original CoCo being used as a terminal with an acoustic modem. How many out there remember doing something like this? I sure don't!

Speech and Sound for OS-9 Level II

Robert Gault

Modifying the Tandy Speech/Sound Cartridge for use with OS-9

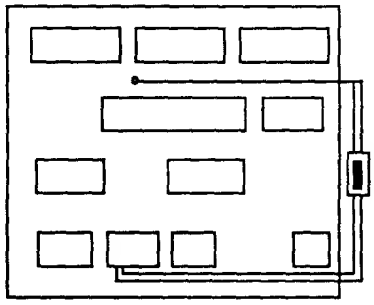
The Tandy Speech/Sound Cartridge (SSC) is an interesting device made for the Color Computer 2 in 1984. The SSC's functions are: 1) convert text into speech, 2) play 3-part musical harmony, 3) create complex sound effects, 4) create speech via an allophone table. The SSC has 8 speech and 8 sound storage buffers. The hardware is based on General Instrument Corporation's GI PIC 7040-510 chip set: PIC 7040, AY3-8913, SP0256-AL2.

I don't intend to review the device as it is no longer in production. The thrust of this article is how to make the device work with OS-9 Level II. Let's start with some history. The Coco2 had available OS-9 Level I version 2. This software ran at .89MHz and expected at most 64K RAM.

The design of the SSC incorporates the above parameters. It expectes a clock signal (Q) of .89MHz which is doubled internally to 1.79MHz and used by the sound effects chip. If the Coco clock is set to 1.79MHz (fast mode), all sounds and tones go up one octave although speech is not affected. The SSC also requires a -6v DC source for it's operational amplifiers. This is created by feeding the Coco E-clock line to a coil and rectifying the output. The circuit used fails at 2MHz because insufficient negative voltage is produced to run the amplifiers.

Hardware Modification

Connect a 470pF ceramic capacitor across the leads of R16. This capacitor value works on my unit. Check the output of the negative supply at both 1MHz and 2MHz. Look for minimal change from -6v at both 1 and 2MHz. Use a value that gets +-1v of the target value. This modifies the negative power supply for 2MHz operation.



SSC Pack

"o" connects to pin17 IC8 & pin20 IC5

Connect a single throw double pole switch with the common lead going to the circuit board feed-through as indicated above. Pole A can go to IC3 pin 8 and pole B to pin 9. Mark the switch "normal" when pin 8 is in circuit and "fast mode" when pin 9 is in circuit.

You must now cut the trace line leaving IC3 at pin 8 on the bottom side of the printed circuit board. BE CAREFUL! This modifies the clock line for 2MHz operation.

The New Software

OS-9 Level I version 2.00.00 did contain an sscpak driver and descriptor but this software was not supplied with OS-9 Level II. This may have been oversight or deliberate as an unmodified SSC won't work at 2MHz. I have looked at the Level I software and found neither the driver nor descriptor conformed to good OS-9 practice. I have written a new driver and descriptor for the SSC and tested it under OS-9 Level II conditions. I have also written a short Basic09 program to demonstrate how the software might be used.

The replacement driver and descriptor for OS-9 Level II are not needed in the strictest sense as it is possible to use the SSC pack with direct PEEKs and POKEs as shown in the owner's manual.

If you are an OS-9 purist, then you will want to use a driver. The new driver has several system services calls that you can use:

GETSTAT

SS.Ready regA = path regB = 1
returns regB = content of $FF7E;
bit 7 0=not ready 1=ready for data
bit 6 0=speaking 1=not speaking
bit 5 0=sounds on 1=no sounds
bits 0-4 N.A.

SETSTAT

SS.Mode regA = path regB = 200
regY = mode
0 = speech mode; chr$($0A) not passed. chr$($0D) will turn on speech
1 = direct access; all values passed. chr$($0A) or chr$($0D) just another byte of data

SS.Audio
regA = path
regB = 201
regY = action;
0 = send soft reset to ssc pack; will stop ALL sound production of any kind; cart. snd line stays active
1 = turns on cartridge sound; used with tones & noise
2 = turn off cartridge sound line; ssc pack may still be active.
Use both 0 & 2 to completely stop all action.

READ

Not a possible attribute of this driver.

WRITE

All data whether text or code is sent to the SSC pack in standard write format. You may PRINT #path, PUT #path, I$Write, or I$WritLn. In fact, you can just LIST a text file to the pack: list filename >/ssc

This driver has been tested in the MultiVue environment and will work with the following caveat. Speech will sound normal but tones will be affected if your screen has an active joystick or mouse. This is unavoidable given the Coco hardware which accesses the DAC and sound function via the same chip.

Attempting to send speech or sound to the pak from more than one window is possible but not very productive. The pak will sound simillar to two people talking at the same time.

One point should be made about the operation of the driver. The normal mode for speech puts the driver to sleep until speech stops. This will cause a shell to freeze in the same manner as when printing a long file. There are two ways to avoid this if you must: 1) run the SSC concurrently, 2) run the SSC in direct mode.

The first (ex. list startup >/ssc&) is the better method. This will result in an extra shell being present for a short time but should otherwise cause no problems. The second method may cause an overflow of the ssc speech buffers. When that happens, the pak will not be able to speak until cleared via 1) a soft reset, 2) $00 stop all action, or 3) $C7 stop all speech.

Driver/Descriptor Installation

It is not necessary to create a new boot file to test the SSC under OS-9. For testing purposes or just to conserve memory (how often will you use the SSC after all), merge the driver and descriptor files together. This will prevent error 237 from occuring when you try to use the SSC. An example follows:

```
merge sscpak.dr ssc.dd >/dd/cmds/ssc
attr /dd/cmds/ssc e pe
load ssc; iniz ssc
```

Basic09 Demonstration Program

Included with this article is a short Basic09 program to demonstrate the new driver. The program will give you a good idea of how the new system service calls can be used.

The SSC will now be ready to dazzle you with it's erudition in philology. If you're nice to it, it might even "tell" you what erudition in philology means.

(listings on page 6)

< 268'm >

SSC Device Descriptor Assembly Source:

```
      nam  SSC
       tl   OS9 Device Descriptor
       ifp1
      use  /dd/defs/os9defs
      endc
TyLg    set  Devic+Objct
AtRv    set  ReEnt+Rev
Rev     set  $02
       mod  Eom, Name, TyLg, AtRv, MgrNam, DrvNam
      fcb  $03 Mode byte; update
      fcb  $07 Extended controller address
      fdb  $FF7D Physical controller address
      fcb  initsize-*-1  initilization table size
      fcb  $00 Device type:0=SCF, 1=RBF, 2=PIPE, 3=SCF
      fcb  $00 Case:0=up&lower, 1=upper only
      fcb  $00 Backspace:0=bsp, 1=bsp then sp & bsp
      fcb  $01 Delete: 0=bsp over line, 1=return
      fcb  $00 Echo:0=no echo
      fcb  $00 Auto Line Feed:0=off
      fcb  $00 End of line null count
      fcb  $00 Pause:0=no end of page pause
      fcb  $00 Lines per page
      fcb  $08 Backspace character
      fcb  $18 Delete line character
      fcb  $0D End of record character
      fcb  $00 End of file character
      fcb  $04 Reprint line character
      fcb  $01 Duplicate last line character
      fcb  $00 Pause character
      fcb  $00 Interrupt character
      fcb  $00 Quit character
      fcb  $00 Backspace echo character
      fcb  $00 Line overflow character (bell)
      fcb  $00 Init value for dev ctl reg
      fcb  $00 Baud rate
      fdb  NAME Copy of descriptor name address
      fcb  $00 ACIA XON char
      fcb  $00 ACIA XOFF char
Initsize equ  *
NAME    equ  *
     FCS  /SSC/
MgrNam  equ  *
     FCS  /SCF/
DrvNam  equ  *
     FCS  /SSCPAK/
      EMOD
EOM     EQU  *
```

SSC Driver Assembly Source:

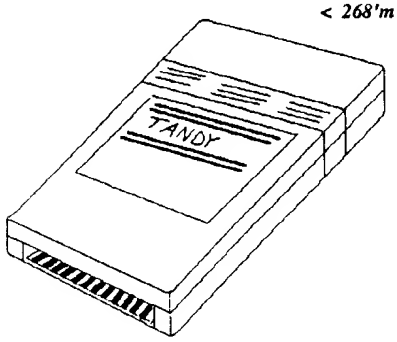
```
* Driver for Tandy Speech / Sound Pak
* Replacement for Tandy SSCPAK driver
*  by Robert Gault; Dec.1,1991
* Code cleaned up; OS-9 rules for device
*  memory area enforced.
* Added timeout error trapping to prevent
*  system lockup if no ssc pack.
* If device not present, driver will report
*  error and time out. This will happen when
*  $0D is sent to ssc.
* Modification History
* Dec 2, 1991  added Getstat and Setstat calls
*        Getstat SS.Ready      Entry A=path#
*           Exit CC clear; B=content of $FF7E
*           Bits are active low.
*  bit 7 clear device not
* ready; busy*
* bit 6 clear if speaking; speak*
* bit 5 clear if sound effect
*  in progress; sound effect*
*    Setstat SS.Mode      Entry
* A=path#; Y: 0=speech;
* 1=direct access
* B=200
* SS.Audio
* Entry A=path#
* Y:1=turn on cart. snd line
* B=201  Y: 0= " off cart. via
*  ssc reset
*  Y: 2=turn off snd line
*  no effect on pak
* Feb 5, 1994  Added more
* F$Sleep calls and timeouts;
* more code cleanup.

SS.Mode equ 200     user defined setstat
service codes
SS.Audio equ 201

* ssc hardware status bits; active low
BUSY equ %10000000 ssc status bits; all
are active low
SPEECH equ %01000000
SOUND  equ %00100000

 nam SSCPAK
 ttl OS9 Device Driver

* ifp1 use defsfile
 ifp1
 use /dd/defs/defsfile
 endc
TyLg set Drivr+Objct
AtRv set ReEnt+Rev
Rev  set $02

 mod Eom,Name,TyLg,AtRv,Start,Size

* Data defined here
 rmb $1D    space required of all drivers
MUX   rmb 2 previous PIA settings storage
PIASND rmb 1  "   "   "   "
sflag rmb 1 signals direct access mode when not
0
Size equ .

 fcb $03 mode byte; =update; required byte for
all drivers

Name equ *
 fcs /SSCPAK/
 fcb 3      edition # used by ident

Start equ *
 lbra init
 lbra read
 lbra write
 lbra getsta
 lbra setsta
 rts              term(inate)

init ldb #$01 send soft reset to ssc pak
 stb [V.PORT,u]
 clrb
 stb [V.PORT,u]
 stb sflag,u default mode is speech
 rts

read ldb #E$BMode illegal mode
 bra error

getsta cmpa #SS.Ready
 bne geterr
 ldx PD.RGS,y get address of caller's stack
 ldy V.PORT,u get base address of port
 leay 1,y point to status/IO byte
 ldb ,y read ssc pack status
 stb R$B,x send raw data to user; reg.B
getrts clrb no os9 errors
 rts
geterr ldb #E$UnkSvc
error coma
 rts

setsta ldx PD.RGS,y get address of caller's
stack
 clr R$B,x clear reg.B
 cmpa #SS.Mode
 bne staudio
 clr sflag,u flag initialize flag for speech
 ldb R$Y+1,x check LSB reg.Y; 0=speech
1=direct access
 beq getrts
 cmpb #1
 bne geterr
 com sflag,u flag now set for direct access
 bra getrts

staudio cmpa #SS.Audio
 bne geterr
 ldb R$Y+1,x check LSB reg.Y; 0=stop ssc
action 1=PIA sound on
 cmpb #1
 beq soundon
 cmpb #2
 beq sndoff
 tstb
 bne geterr
 lda #1
 sta [V.PORT,u]
 clra
 sta [V.PORT,u]
 ldy V.PORT,u
 leay 1,y
 lbra send
sndoff ldx #$ff00
 lda $23,x
 anda #^8 turn of sound bit
 sta $23,x
 bra getrts
soundon ldx #$ff00 turn on cart. sound; point
to base address of Coco I/O
 lda 1,x get current MUX values
 ldb 3,x
 anda #^8 set for cartridge sound source; ie. ssc
pak
 orb #8
 sta 1,x set new MUX values
 stb 3,x
```

```
 lda $23,x
 ora #8 enable sound
 sta $23,x
 bra getrts

write ldx #$ff00 PIA base address
 ldy V.PORT,u point to ssc port
 leay 1,y point to I/O byte
 tst sflag,u are we in direct mode?
 bne send if direct mode go to send
 cmpa #C$LF linefeed? if so do not send value
 beq exit
 cmpa #C$CR carriage return
 bne send     any other code to PAK buffer
 pshs cc
 orcc #$50 kill interrupts while reading PIA
settings
 lda 1,x get MUX data
 ldb 3,x
 std MUX,u save current setting
 anda #^8 set for cartridge input
 orb #8
 sta 1,x tell MUX
 stb 3,x
 lda $23,x get sound line data
 sta PIASND,u save current setting
 ora #8 enable sound
 sta $23,x tell PIA
 puls cc  reset/restore interrupts
 lda #C$CR
 bsr tstssc wait for pack to be unbusy
 bcs wrterr pack timed out; error
 sta ,y tell PAK to speak; send CR
 bsr ack wait for pack to start speaking
 bcs wrterr timed out without speaking; error
 bsr soundx wait for pack to stop speaking
 bsr rstPIA reset sound lines
exit clrb
 rts

wrterr bsr rstPIA reset sound lines
generr coma
 rts

rstPIA pshs b
 ldd MUX,u get previous setting
 sta 1,x reset MUX
 stb 3,x
 lda PIASND,u get previous setting
 sta $23,x reset sound line
 puls b,pc

send bsr tstssc wait for pack to be unbusy
 bcs generr
 sta ,y send character
 rts

* Sleep until ssc is ready or times out.
tstssc pshs x,u
 ldu #$1000
tstloop ldb ,y get status
 andb #BUSY
 bne ok return if not busy
 ldx #1
 os9 F$Sleep sleep for the rest of the tick
 bcs slperr
 leau -1,u update timeout count
 cmpu #0
 bne tstloop
pakerr ldb #E$NotRdy
 orcc #1
slperr puls x,u,pc
ok clrb
 puls x,u,pc

* Sleep until ssc starts talking or times out.
ack pshs x,u     acknowledge
 ldu #$1000
ackloop ldb ,y check ssc pak
 andb #SPEECH
 beq ok
 ldx #1
 os9 F$Sleep
 bcs slperr
 leau -1,u update timeout count
 cmpu #0
 beq pakerr if device does not start speaking
then device not ready
 bra ackloop

* Blow-hard ssc might talk forever so can't
time out; sleep through speech.
soundx pshs x,u
sloop ldb,y
 andb #SPEECH
 bne ok go if finished
 ldx #3
 os9 F$Sleep
 bcs slperr
 bra sloop

 EMOD
Eom equ *
```

Basic09 SSC Demo Program:

```
PROCEDURE ssc_demo
TYPE        registers=cc,a,b,dp:BYTE;
x,y,u:INTEGER
DIM regs:registers
DIM i:INTEGER; path:BYTE
DIM aa,bb:STRING[255]
PRINT CHR$($0C);
aa:=""
(* Direct sound register access; channel-A
freq.; mix channel-A *)
(* volume fixed mdeium amplitude *)
bb:=CHR$($AF)+CHR$(0)+CHR$(50)+
CHR$(7)+CHR$(62)+CHR$(8)+CHR$(12)
OPEN #path,"/ssc":UPDATE
regs.a:=path
LOOP
PRINT "Select an option."
PRINT "(1) Getstat"
PRINT "(2) Setstat: flag speech"
PRINT "(3)       : flag direct input"
PRINT "(4)       : turn on audio"
PRINT "(5)       : turn off ssc pak"
PRINT "(6)       : turn off sound line"
PRINT "(7) Send text; via translator"
PRINT "(8) Send sound; via direct register
access"
INPUT ">>",aa
IF aa>"8" OR aa<"1" THEN
CLOSE #path
END
ENDIF
i:=VAL(aa)
PRINT CHR$($0C);
ON i GOSUB 1,2,3,4,5,6,7,8
ENDLOOP
1 regs.b:=1
RUN syscall($8D,regs)
IF LAND(regs.cc,1)<>1 THEN
PRINT USING "'$FF7E bit7=',i2,' bit6=',i2,'
bit5=',i2",LAND(regs.b,128)/
128,LAND(regs.b,64)/64,LAND(regs.b,32)/
32
ENDIF
RETURN
2 regs.b:=200
regs.y:=0
RUN syscall($8E,regs)
GOSUB 10
RETURN
3 regs.b:=200
regs.y:=1
RUN syscall($8E,regs)
GOSUB 10
RETURN
4 regs.b:=201
regs.y:=1
RUN syscall($8E,regs)
GOSUB 10
RETURN
5 regs.b:=201
regs.y:=0
RUN syscall($8E,regs)
GOSUB 10
RETURN
6 regs.b:=201
regs.y:=2
RUN syscall($8E,regs)
GOSUB 10
RETURN
7
INPUT "text? ",aa
PRINT #path,aa
PRINT
RETURN
8
PRINT #path,bb;
RETURN
10 IF LAND(regs.cc,1)=1 THEN
PRINT "Error "; regs.b
ENDIF
RETURN
```

< 268'm >



Announcing

5th Annual Atlanta CoCofest

Holiday Inn, Northlake

October 1 & 2, 1994

Show Hours:
Sat., Oct 1 9:00AM-5:00PM
Sun., Oct 2 9:00AM-3:00PM

Vendor Setup:
Fri., Sept 30 6:00PM-9:00PM
Sat, Oct 1 8:00AM-8:45AM

Admission:
$10.00 (entire show)

Reservations:
Holiday Inn Northlake
800-465-4329
or
404-938-1026

Sponsored by:

Atlanta Computer Society
PO Box 80694
Atlanta, GA 30366
BBS: 404-636-2991

CoCo Hard Drives

F.G. Swygert

Using a hard drive with Disk BASIC.

Using a hard drive with OS-9 is pretty straight forward once the physical connections are made. Simply load the drivers and descriptors and go! Well, that is a little overly simplified, but OS-9 was designed to use a hard drive. The multiple directory structure can easily handle large media with no problems at all.

Disk Extended Color BASIC (DECB) is different... *much* different! The first "problem" is that it isn't an operating system at all... there is nos such thing as "RS-DOS". A "DOS" is a Disk Operating System. DECB gets this misnomer because it does operate disk drives, but only in a primitive manner.

A true DOS has multiple directories for handling different sizes of media. It also has the capability of adding drivers and external commands for increased flexibility. DECB has none of these or many other capabilities of a true DOS (these two are mentioned because they are the main reasons DECB doesn't handle hard drives natively).

DECB is a BASIC programming language. Extended Color BASIC was modified to allow the simple use of disk drives, thus becoming Disk Extended Color BASIC. In order to keep both the size and complexity of the language to a minimum, only one directory per disk was allowed. A method of using external drivers was also left out. This made the system simple and easy to use, but also restricted the size of the disk meda to that burned in the ROM.

It didn't take long for DECB users to figure out how to modify the system and add more capabilities. The most famous of these modifications is Art Flexser's ADOS series. Art's enhancements allow using up to 80 track drives (720K), but only one side at a time.

In the mid eighties, when hard drive prices started falling under $500.00, several systems were developed to add hard drive capability to DECB. The two most popular systems were Burke & Burke's Hyper I/O and RGB Systems' RGB-DOS. Both handle the problem of allowing DECB access to a hard drive in very different ways.

Hyper I/O uses "mass storage areas" (MSAs). These areas can be of any size between 2K and 3MB. Instead of drive numbers, "drive handles" are used to name the MSAs. Only four "handles" can be active at any one time even though there are many MSAs. A special "open drive" command activates a particular MSA, while the "unload" command closes it.

There two types of MSAs, "flat" and "RBF". Flat MSAs can take up an entire drive. RBF MSAs are only allowed a portion of a drive. RBF MSAs are also compatible with OS-9, meaning that any files stored in a DECB RBF MSA can be copied, deleted, or a directory displayed from OS-9 using three new commands: HCOPY, HDEL, and HDIR.

The way the MSAs interoperate make it easier to share files between DECB and OS-9 portions of the drive. The size of the MSAs can be adjusted at any time.

Hyper I/O is reset protected, so it can be run from disk. The problem with running from disk rather than burning into an EPROM is that almost all machine language programs will return to DECB upon exit. Hyper I/O would have to be reloaded from disk.

If you don't expect to be copying files between the OS-9 and DECB portions of the hard drive very often, RGB-DOS may be the easier way to go (there are utilities that allow exchange of files between OS-9 and DECB floppies). RGB-DOS is a patch to DECB that allows up to 256 drive numbers (0-255). Each of these "drives" are actually 156K (35 track single sided disk capacity) partitions on the hard drive. The advantage to this system is its ease of use. Simply type DIR 100 to get a directory of that partition. All other commands are also patched for the higher drive numbers.

But what about the floppy drives? A special DRIVE ON/OFF command moves the first four drive numbers (0-3) between the hard and floppy drives. DRIVE ON activates hard drive partitions 0-3, while DRIVE OFF disables the hard drive partitions. DRIVE ON is the default. Since some DECB programs are hard coded to run only from the first four drive numbers, these numbers can be assigned to any other partition.

When first started, RGB-DOS looks for the program "AUTOEXEC.BAS" on the first hard drive partition. If found, this program is run. Most RGB-DOS users will create a BASIC menu and name it AUTOEXEC.BAS so that any program can be easily run. Even if a machine language program resets the computer when exited the menu will come back up.

A feature useful for auto-start programs and OS-9 is built into the DOS command. Typing "DOS 23" will execute the DOS command on drive partition 23. Any program that starts with the DOS command residing on drive 23 would then be executed, including OS-9 Boot. Even if the hard drive will be used exclusively for OS-9, it is a good idea to create at least six DECB partions. Not only can a few DECB utilities or programs be loaded, but six different boots could be easily set up as well!

RGB-DOS expects a drive capable of formatting 256 byte sectors. The current version is designed for SCSI interface drives only. The only system it is guaranteed to work with is the Ken-Ton SCSI controller and a Seagate "N" series drive with ROM revision 104 or greater (there is an "N" on the end of the drive designation... ST251N; the ROM revision is usually printed on the ROM chip, which may be a square surface mount device. Any revision number over 104 will support 256 byte sectors). It does work with other SCSI systems and drives, but the manufacturer does not guarantee it to. The only other SCSI drives known by the author to support 256 byte sectors are Rodime 65x series (where "x" is any single digit number).

RGB-DOS will work with PC type 512 byte sector drives. The problem is that you lose half the capacity of each sector as RGB-DOS ignores the extra 256 bytes. If you only want to have a few DECB partitions on a predominantly OS-9 drive, this is no real problem. Six 156K partitions would take up approximately one megabyte of drive space. On a 20MB or larger drive, 1MB of DCB space shouldn't be that noticeable, and may be well worth the sacrifice to get those couple DECB utilities or applications; or six different OS-9 boot files, on a hard drive. Using a 512 byte sector drive exclusively with DECB just isn't worth it unless a big drive can be found at a great price.

RGB-DOS will only support up to a 70MB drive under DECB. With the Ken-Ton interface, however, up to seven hard drives can be installed.

Is a hard drive in your DECB future? If you use it exclusively for DECB the cost probably isn't worth it. If you intend to split the drive between DECB and OS-9 use however, it definately is.

Which system is right for you? If your predominant need is for ease of use under DECB, then you need RGB-DOS. When exchange of files between DECB and OS-9 is most important, then Hyper I/O is clearly the winner.

RGB-DOS is available from:
FARNA Systems
Hyper I/O is available from:
Burke & Burke

< 268'm >

The Hardware Hacker

Dr. Marty Goodman

Hacking with Small Systems...getting by with fewer I/O lines.

This last month I began serious work on a project I'd been meaning to do for a number of years. I have a crude prototype more or less running as I type up this column. The project involves a CoCo, and illustrates some general principles of small, dedicated systems design.

My project is building a device that can measure the actual charge capacity of small to moderate size lead acid and nicad battery packs. The packs I wished to be able measure varied from 1.2 to 12 volts and from 1 to 10 amp hours in rated charge capacity. I use numerous such packs both in amateur radio applications, and as the power source for powerful bicycle lights for serious (30 to 100 mile long, in pitch darkness, including 35 to 40 mph descents on steep hills) night riding.

To do this, I had the computer reading a digital volt meter, and keeping time, as a (previously fully charged) battery pack is discharged thru a known load resistor. The computer samples the voltage of the battery every second, and every 5 minutes it calculates the number of amp hours drained during the last 5 minutes. When the battery voltage drops below a pre-set point, the test is ended and the battery automatically hooked back up to a charger. Amps hours are calculated by dividing the measured voltage by the value of the load resistor, then dividing that by 12 (for 5 minutes, the sampling interval, is 1/12 of an hour). I can choose different values of load resistor for different voltage and capacity battery packs, and of course set the end-of-test voltage accordingly. I was using a Green Mountain Micro parallel port card as my I/O card for this project. This consists of a single 6821 PIA hooked to the CoCo bus, providing 16 lines of bi-directional I/O.

The problem I encountered was that my device seemed to need more than 16 lines. It took eight lines to read the four digits of the volt meter, for the meter provided four bits of BCD information for the value of a given digit, and had four strobe lines that selected which digit was being displayed. I needed to monitor all eight of these. I then needed to display hours, minutes, and seconds on 7 segment LEDs, and display the cumulative amp hours (four digits of information) as well. Not to mention lines for status lights, action buttons, and telling the computer what value resistor I was using. It seemed at first like I'd need a zillion lines to do all this.

Fortunately, I'd recently had a long chat with my old friend Steve Bjork, who is currently in the forefront of programing modern home computer games for commercial games systems. Steve was telling me of the trend in small, embedded-controller systems design of using multiplexing and serial data transfer quite intensively to get around the problem that modern controller chips tend to have relative few physical I/O lines.

The current generation of embedded control systems have the processor talking to RAM, ROM, and external devices all via two or three lines for each, using serial instead of parallel data transfer. One can, for example, read a serial ROM using a data line, a clock line to clock out the bits, and a reset line to bring the address counter back to zero when one needs to. This greatly reduces board and chip size for a given embedded system.

There is a small to moderate price to pay in terms of things happening at slower speeds overall, and in terms of somewhat more complex demands on the programer when talking to things, but this is in a large number of applications completely overshadowed by the great savings in size and cost of the system. I decided to apply these principles to my design, for even tho the controller I was using (a CoCo 2) was not quite a modern 18 pin PIC controller device, I was facing similar design problems of minimizing I/O lines.

Instead of using separate 7 segment displays and drivers, which would have required at least fourteen I/O lines for the 10 digits I needed to display (four lines to specify a digit, and ten lines to select a given digit), I used counter modules instead. One such module that I used to display the cumulative amp hours was a four bit counter, which had a clock line and a reset line. Each time I updated the amp hour count (every five minutes) I had only to wiggle the clock line high and low as many times as the number I wished to add to the count. Since both the counter and the CoCo were capable of handing clock rates of 100K Hz, I could advance the four digit counter any given amount in a virtually invisible fraction of a second. Thus, using only a clock and a reset line, I'd gotten control of one four digit display.

Next I used a four digit counter to show minutes and seconds. I had the counter's clock get a pulse every second. Then I had a problem: When the counter changed from 59 to 60 in the lowest two digits (seconds digits), I needed to reset those lowest two digits to 00 and advance the third digit (ones of minutes digit) by one. But I had only a plain four digit decimal counter. The solution: Merely program the CoCo to blast out rapidly 41 pulses (instead of one pulse) to the counter's clock whenever the seconds count changes from 59 to 60. This both sends the lowest two digits back to 00, and advances the third (minutes) digit. Add to that code that resets the entire counter whenever the minutes count changes from 59 to 60 (using the reset line on the counter), and one is able to use an ordinary decimal counter to display seconds and minutes quite nicely. In the actual working display, one absolutely CANNOT see the individual rapid counts when the clock changes from displaying 59 seconds to displaying 1 minute 00 seconds. The transition appears perfectly smooth.

In the end, using three separate counters, I was able to handle a display of twelve digits of information (four digits of hours, two of minutes, two of seconds, and four digits of amp hour display) using only five I/O lines (for I could use a single reset on two of the counters in this particular implementation).

To further economize in the final design, I may use a 74LS157 4 bit multiplexer chip between the volt meter and the computer. This would allow me to go from 8 lines of I/O to 5 lines of I/O dedicated to reading the volt meter, for I would divide the 8 bits into two groups of four bits, and use a fifth bit to select which of the two groups of four the PIA was looking at. Of course, in a mass production design for such a device, one might go further and use a means of measuring voltage that could talk serially to the computer, and thus have the volt meter communicate to the computer using only three or four lines (reset, clock, and data, data ready).

I plan on employing one other little trick to decrease the number of needed PIA lines in this, my prototype design. I need to be able to "tell" the program which of four values of load resistor I'll be using, and have the program accordingly alter a constant it uses when calculating amp hours. Normally one would have the information entered via PIA input lines connected to switches or buttons. In this case, that of a one of a kind prototype and a situation where I'm running low on available PIA lines, I may employ another approach: I may put multiple copies of the program into the EPROM in the device, and select WHICH copy I'm using by switching the status of the high order address lines of the EPROM. This trick has been used for years by CoCo owners who wished to switch between ADOS and DECB. They'd burn copies of both into the two halves of 27128 (16K by 8) EPROM, then put a switch on the A13 (high order address line) that would set

(continued on page 12)

Operating System -Nine

Rick Ulland

Backups, C compiler tips, and graphing with DynaCalc and PhantomGraph

There's a new shell in town, and it's name is 2.2a. Or so I see in the other window. Curtis Boyle fixed up the command line history version of shell+, so of course it's a feature of the new Nitros9 release. He was also nice enough to release a 6809 version, complete with one of the traditional ipc files....but whats this? A file called simply 'shell'. An actual executable code thing remarkably similar to an OS-9 module. A refreshing change, if the lawyers stay away- the patch files are getting longer than the original code. Last seen on compuserve.....got to install this thing.

Perhaps moving this far north has affected my senses, but it's already fall here, near as I can tell. Between school and the winter programming season, the coco gets a new workout, and lots of data gets shuffled around. Which is a long-winded way of asking if you have backed up your hard drive lately. Time for a repack as well! If you don't have and can't afford a repack utility, restoring a streaming backup to a fresh format is an extreme measures way to do the same thing.

When I started out, I used the utility that came with my hard disk. Seemed to make sense at the time, and it was free. But it was useless for anything but an identical reproduction of the original drive. Now I have stream. It's not only 10 times faster, but allows parts of an archive to be extracted- if you had to, you could set your system back up on floppies with just the archive and a boot disk. And it's only $25, shareware. With a name like Bruce Isted (the programmer), it's got to be good.

Floppies also suffer from fragmentation, and if you run separate cmds and data drives, the old cmds disk might be why you are getting impatient more often. Even though there is little free space to repack, a dsave /d0 /d1 ! shell will rewrite each file as a contiguous block. But then Alice.....

Microware/Tandy C Compiler Hints

Ask some folks what an operating system is and they'll tell you it's the code needed to properly support a C compiler. And they do have a point, many applications began with a curly brace.

If you're lucky, you've found a copy of Microware's Ancient and Honorable C compiler. Tandy will be happy to sell you a Software Assembly version (#26-3038) complete with the now traditional badly Xeroxed pile of loose pages manual- even worse than the original, which itself looked like a Xerox of a mimeograph of some typewritten pages and instantly unbound into loose sheets once opened.

If you've ever tried compiling a program according to the manual, using the supplied modules, you just might be doubting your luck. Like much of the 'serious' OS9/6809 software, this puppy is old- still bearing a v1.00.00 stamp after a dozen years. Of course it's been improved, just not by Tandy.

The first step is improving the quality of code produced. A large percentage of a compiled program is snippets of prewritten code stored in a 'library', and the c compiler includes one, know as stdlib. It was OK for it's day, but that day has passed. The upside is it can be replaced with either of Carl Kredier's versions- the full featured clib.t includes transcendental math functions while clib gives up the complex math to make the final program smaller. And best, Carl's libs come with their own u-printem docs, replacing most of Tandy's smudged pages.

The old C does graphics as well, but Tandy strewed clib to the winds- the library itself was in the DevPak, some of the manual pages in the MultiVue binder- one had to buy everything! Or obtain Mike Sweet's cgfx7, which is not only nicer than stock, but will use up the other half of your new printer ribbon to produce more manual.

It's possible to use the stock compiler and these libraries to write code as good as anybodies, but the compiler is still klunky to use. The main operational problem with this compiler is speed. Besides the normal problem of a 6x09 with too much to do, it was designed to work even on Level One systems, and so is broken down into smaller pieces such a machine can digest. Communication between all the pieces is done by writing work files- and hours can go by while floppies churn madly.

Since level Two has the ram, the obvious solution is a ramdisk. Copy your source to the RAMDisk and chd to it. Now all those temp files can be quickly saved to RAM. There is an alternative cc (compiler executive which controls the whole process) that defaults to Ramdisk without the dancing. And of course, loading the whole compiler into RAM helps, especially if the first compile goes badly (don't they always?)

You'll want to move chx anyway, so the output file doesn't end up in main CMDS. A short shell script will do all of this for you so you can get right to work.

Although we don't really get into programming here, a few notes are in order. First, ANSIfront. Vaughn Cato's little c preprocessor does a pretty good job of turning ansi c straight out of your TurboC textbook into something the old compiler can handle. This is a very big deal if you are considering a port from another machine. There is a companion c.prep (Jim McDowell) to read the weird looking source and spruce it up for ANSIfront.

There are still good reasons to learn the K&R way of doing things, even (perhaps especially) if the coco is a temporary condition. Microware is extremely proud of their ANSI C package, ensuring the older version will remain popular with OSK resellers. You, of course, will be one of the few required to port a K&R style program to ANSI.

It's difficult to even find a reference for K&R C. They are impossible to locate, and expensive once found. Public Domain to the rescue! The uncompleted Hitchhiker's Guide to C is a good beginning, but ends just as things are getting interesting. There is a more tomb-like tutorial also available, and this one lasts to an end- I'd still use HGtC as far as it goes.

Another source of C info is magazines. "The OS-9 Underground" leans quite a way towards being a C magazine, although some of the code isn't exactly beginner level stuff. Worth a look if you are interested in the language. This magazine is a good place to pick up pointers as well- just watch out for Joel, he's a sneaky one! (ducking).

C wins the disk of the month award. If you don't have all this stuff, a disk is only $5.

Graphing Dynacalc

Dynacalc makes a great little spreadsheet, but it's graphing function leaves a bit to be desired. Tandy had the solution in PhantomGraph (PG), but presented it in the usual Tandy manner, in otherwords you might have found one buried under 37 copies of Zone Runner, if you looked. PG comes with a Tandy manual.

Before jumping in, a quick review of PG's data structure. The overall structure is four groups of twenty elements. Think of a group as a section of a stacked bar chart. Each bar has up to four sections, and there may be twenty of these bars. Groups can also be graphed side by side in a bar chart- 80 bars per chart. Each element may contain 2 data cells. On scatter plots the second cell is used as the x value. Some graphs (for example pie charts) ignore this cell. In addition, each element has three characteristics- (color, pattern, and attribute), and a title.

The secret to using PG is to never enter any data directly into the program- it's not really set up for that. Instead, PG should be force fed data directly from dynacalc.

Operating System - Nine

(continued from page 11)

Unless your spreadsheet is very small, set aside a block of the it for PhantomGraph data. This area also serves as a text summary screen- leaving the label mid line allows ignoring the graphing data while crunching the numbers. It might look like this-

pat1 pat2 pat3 pat4 label1 data1 data2 data3 data4

Use dynacalc's replicate function to set up the pattern table quickly- for example put the formula (d12+1) in cell e12, then replicate relative down the column. Other PG information like colors and attributes can be added to the pattern tables if needed, but keep it fairly simple, all these row numbers have to be entered in PG!

The data cells may just be equates to cells in the main sheet, or do quite a bit of math in their own right. Using /s#s and /s#l, this block of the sheet can be saved out as a separate file. To graph a series of similar data, this s# file can be plopped in for a quick recalc, then saved again (the new data) for conversion (save in column order for graphing).

Inside PhantomGraph itself, utilities/convert/dyna pops up the main entry window. After the obvious file naming, the columns containing the first data and pattern cells are input. Click on the window, but not on a selection, to save this group. When the window comes back, an addition group of data can be saved by changing the group number and column info. Click on the window again, then append. Once all groups are converted, click off the window to stop. You can reenter this file at any time by using it's name with dyna again.

With the data file saved, files/open/data gets it into PG itself. After setting up verbosity (titles and such), files/close saves a graph file version- these graph files save the added info. And the rest is pretty much like the tutorial supplied with PG.

And so.....on to Atlanta!

I do hope to be at the 'fest this year, if the work schedule (and funds!) permit...

< 268'm >

Comments and questions may be sent in care of 68'Micros or directly to Rick at:
Rick Ulland
449 South 90th
West Allis, WI 53214
E-mail is rickuland@delphi.com

The Hardware Hacker

(continued from page 10)

which 8K block of data was seen by the CoCo as the disk ROM. In this case, my program will be a bit over 1K in size before it's done. Let's round that up to 2K. Fine. I can then use a 2764 EPROM (8K by 8) and have FOUR copies of the program in that EPROM, each using different load resistor constants. By switching the status of the A12 and A11 lines of the 2764, I can select which 2K block of code is being seen by the computer. Thus, I can select among four versions of the same program without using ANY PIA lines. Indeed, if I chose to use a 27512 EPROM (which, like the 2764, is a 28 pin part and which, like the 2764, is quite inexpensive on the surplus market, going for about $2 each) I could select among 32 versions of the same 2K program!

I may next month bring this project over to run on the dedicated 6809 controller card I referred to some months ago, and hope at that time to report on that quite promising-looking little card.

On other matters, I recently had a report from a Delphi member concerning a bizarre incompatibility between a particular model of Epson 24 pin printer (I forget the exact model number) and his serial to parallel converter box. Seems this fellow found the printer would work just fine with a PC compatible's parallel port, and that four other brands of parallel printers (including Panasonic and Cannon ink jet) would work just fine with his CoCo via the serial to parallel device. But when this particular model of Epson was hooked up to the serial to parallel converter, and he told the CoCo to print, all he got was garbage. Whatever the problem was, it did NOT involve a bit being stuck. Indeed, the garbage LOOKED as if one had a baud rate problem between the S/P converter and the computer, tho this was not the case. After further investigation, it seemed as if that particular model of printer (a recent model 24 pin printer) simply had somewhat odd timing for its parallel port acceptance of data, such that it just would not work with that particular serial to parallel converter *(editor: I believe the convertor being used was an old Botek unit... like mine! Other units, particularly those made for PCs, may not have the same problem.)*

< 268'm >

Comments and questions may be sent in care of 68'Micros or directly to Dr. Goodman at:
1633 Bayo Vista Avenue
San Pablo, CA 94806
E-mail: martygoodman@delphi.com

Programming in C

(continued from page 19)

REMEMBER: To pass the function sam(a,b,c) as an argument to another function george(sam,x,y), then include the declaration float (*sam)() (make this declaration within george()) and use it (within george()) as (*sam)(a,b,c). If sam() returns an int or char then (of course) it should be declared as int (*sam)() or char (*sam)()!

```
A root of x=f1(x) is 1.895475     Here's
A root of x=f2(x) is 1.333324     our
A root of x=f3(x) is 1.618026     output.
```

... and (because we use only float and not double, and we gave an error specification of .00005) we get (roughly) 4 decimal place accuracy.

Well, the programming ain't too sexy (how useful are these 3 built-in functions, f1(x), f2(x) and f3(x) ?) and the mathematics is even worse (you can't guarantee that the program won't get stuck in the solve() function ... forever trying to reduce a growing error!), BUT ... we get the idea ... right?

P.J.Ponzo
Dept. of Applied Math
Univ. of Waterloo
Ontario N2L 3G1

< 268'm >

OS-9/OSK Answers!

Joel Mathew Hegberg

Plumbing with OS-9 (pipes); an undocumented mono play command for the MM/1.

Editor's Note: Most of you probably noticed that page 24 was actually page 23! Also, this column contained a note at the end stating that it was continued on page 22. t was actually continued on page 24, the missing page! We begin this issue's column with the missing portion. Sorry for the inconvenience! Listing #4 is on last issue's "microdisk".

Listing #4: playsndm.c

```
=====================
#include <stdio.h>
#include <modes.h>
#include <sound.h>

#define STDOUT 1
extern unsigned char *malloc();

unsigned char *sound,*ptr1,*ptr2;
int size;

main(argc,argv)
int argc;
char *argv[];
{
  int dpath,sample_rate;

  if (argc!=2 && argc!=3)
  {
     fprintf(stderr,"PlaySndm <sndfile>
```

(continued on page 22)

OS-9/OSK Answers!

(continued from page 18)

```
{sample_rate}\n");
        fprintf(stderr," If sample_rate is
omitted, 16000 Hz is used.\n");
     exit(0);
  }

  if (argc==3)
  {
    sample_rate=atoi(argv[2]);
              if (sample_rate<4000 ||
sample_rate>32000)
     {
                fprintf(stderr,"Sample_rate
invalid!\n");
       exit(0);
     }
  }
  else sample_rate=16000; /* default */

  dpath=open(argv[1],S_IREAD);
  size=(_gs_size(dpath)-40)*2;
  sound=malloc(size);
  if (sound==(unsigned char *)NULL)
  {
        fprintf(stderr,"Can't allocate
memory!\n");
     exit(0);
  }
  lseek(dpath,40,0);
  read(dpath,sound,size/2);
  close(dpath);
          for (ptr1=sound+size/2-1,
ptr2=sound+size-2; ptr1>=sound; ptr1--
,ptr2-=2)
      *ptr2=*(ptr2+1)=*ptr1;
 _ss_play(STDOUT,sound,size,sample_rate,
SND_NOSIG, 0);
  free(sound);
}
```

So, what is the second method of playing mono sound samples? Very simple, actually. Andrzej Kotanski discovered that the MM/1's sound driver actually has an undocumented mono-play mode. Actually, it was partially documented but was rumored not to have been implemented... and the partial documentation was quite vague. But, it's extremely easy. You simply need to set bit #0 of the "option flag" in the _ss_play() call. So, you can take Listing #3 and make the following single change and it will play files in mono-play mode.

Change the following line in Listing #3:

```
 _ss_play(STDOUT,sound,size,sample_rate,
SND_NOSIG,0);
```

to read:

```
 _ss_play(STDOUT,sound,size,sample_rate,
SND_NOSIG|0x0001,0);
```

The advantage of using this technique is there's no mono-to-stereo conversion needed (saves CPU time), and your memory buffer does not have to be twice the size of the sound sample. The downside to this is mono sounds played with this technique seem to have more distortion than mono sounds which have been converted to stereo and then played.

And now the "regular feature"...

September is a great month! Okay, I was born in September, but Autumn is a great time of year — not too hot like Summer, not too cold like Winter, not too muddy like Spring, but just right and colorful. And the nights are longer! It seems most computer programmers tend to enjoy computing late at night. So remember during this lovely Fall season, be sure to keep your 68' micros subscription up-to-date, so you'll have plenty of interesting computing sessions in the months to come!

Last issue we talked a bit about named pipes, and how they could be used for interprocess communication. Another great use for named pipes is a temporary data storage, similar to a RAMdisk. Why not just use a ramdisk? Because not everyone has a ramdisk installed on their system. Pipes, on the other hand, are used by the OS-9 Shell and many other system utilities, so pipes almost have to be installed on everyone's OS-9 system.

One possible use of this "named pipe" storage area is an installation program. Many installation programs copy programs directly from one disk to another. This is great if you have more than one floppy or a floppy and a hard drive, but what about those few people that only have one floppy? If this is the case, why not copy all or some of the data to a named pipe and allow the user to swap disks, and copy the data to the second floppy disk? Sounds easy, but is it? Yes, but proceed with caution...

It's very simple to copy files directly to a named pipe. Your installation program may just use the OS-9 "copy" utility to "copy * -w=/pipe" (copies all files in the current directory to named pipes), then after the user swaps disks, use "copy /pipe/* -w=/d0" (copy files in the pipe to drive /d0). The "copy" utility actually looks at the size of the file it's copying, and sets the initial size of the named pipe, so write-blocking won't occur. This would work, but there are a couple problems. First, named pipes are used by a few programs, such as our "headlines" program in the last issue. How do you know named pipes for other programs do not exist, as they would be copied by referencing "/pipe/*" in the copy command (You can use "dir /pipe" to see if there are any named pipe files currently on your system.)? Once a pipe file is read, it disappears... and that could really confuse another program, not to mention you're copying files to the user's disk that aren't part of your installation.

So how about specifically naming your pipes, such as "copy program /pipe/program" and then "copy datafile /pipe/datafile" and then "copy script /pipe/script", etc., and then after the user swaps disks, "copy /pipe/program /d0/program", etc.? That is much better, and should solve the program I noted above, but there's something you may want to know about pipes, and my thanks to John Wainwright for bringing this to my attention. John noticed some of his files actually grew in size by a few bytes after being copied to a named pipe and

then back. What's going on? I used the "dump" utility to examine a file copied in this fashion, and noticed a few null-characters were added at the end of the file. I also noted the "dump" display, which displays 16 bytes per line, did not show any odd number of bytes for the final line. I remembered reading about this in the book, "The OS-9 Guru" by Paul S. Dayan (a book no OS-9 programmer should be without), so let me quote a brief portion for you...

"The actual size of the pipe buffer allocated may be greater than the requested size — the request is rounded up to the nearest multiple of the minimum allocatable block size (16 bytes)."

That explains the additional bytes, and why the "dump" output did not have an odd line at the end. In all the cases I've tried, the appended bytes have always been nulls (value $00), but I have found no official documentation stating this is guaranteed to be the case.

Well that's kind of annoying. In many cases the added null-bytes don't affect anything, but in some cases it certainly would. It looks like we need a more intelligent installation program! One that will store the exact size of each file, to prevent any padding from taking place. Let's store all of the to be installed on the user's disk in a single file ahead of time, which we can call an "install file" or "i-file". We need to define the i-file format, so let's make something up that makes sense...

The first 5 characters of the file should be "IFILE", so we can detect if the file is a valid i-file or not. Next, we'll store 29 bytes for the null-terminated filename, followed by an integer (4 bytes in OSK) for the exact size of the file stored within the i-file. The actual file contents then follows ('size' bytes long). Then, the next file is stored beginning with the 29 bytes for the filename again. If the first character of the filename field is a null ($00), there are no more files in the i-file. There, that was simple. It is important to keep in mind this is merely an example. If you were writing an installation program for commercial-quality software, you would want to have longer pathnames than 28 characters (to specify various directories) and record file attributes so executable files would run properly.

Listing #1 gives a program (make_ifile) to create an i-file and copy files into the i-file. This would be done once and stored to the original diskette. When the user runs the installation program, the i-file should be copied over to a named pipe file, so the user can swap disks if necessary. A sample command-line would be:

```
make_ifile test.ifile file1 file2 file3
  or
make_ifile test2.ifile *.txt
```

Listing #2 gives a program (install_ifile) to copy all of the files contained within an i-file to a given directory. A sample command-line would be:

```
install_ifile /pipe/test.ifile /d0
  or
install_ifile /pipe/test2.ifile /dd/CMDS
```

Installation of new software really can be an easy process if the programmer puts in just a little more effort. Named pipes is just one technique able to be used by a good installation program. You can see the source code listings are not very long, so there truly is little excuse for not supporting easy installation.

Listing #1: make_ifile.c

```
=========================
#include <stdio.h>
#include <modes.h>
#include <errno.h>

#define BUFFSIZE 8192

main(argc,argv)
int argc;
char *argv[];
{
      int ipath,path,size,numbytes,t;

    char *data;

    if (argc<3)
    {
          fprintf(stderr,"make_ifile <ifile>
<file> {file ...}\n");
          exit(0);
    }

    data=(char *)malloc(BUFFSIZE);
    if (data==(char *)NULL)
    {
          fprintf(stderr,"make_ifile: Can't
allocate buffer memory!\n");
          exit(errno);
    }

    ipath=creat(argv[1],S_IWRITE);
    if (ipath==-1)
    {

          free(data);
          fprintf(stderr,"make_ifile: Error
opening file %s for writing!\n
",argv[1]);
          exit(errno);
    }
     write(ipath,"IFILE",5);

    for(t=2;t<argc;t++)
    {
                printf("copying %s to
ifile...\n",argv[t]);
          fflush(stdout);
          path=open(argv[t],S_IREAD);
          if (path==-1)
          {
               free(data);
               close(ipath);
                 fprintf(stderr,"make_ifile:
Error opening file %s for re
ading!\n",argv[t]);
               exit(errno);
          }


        /* record filename */
          sprintf(data,"%s",argv[t]);
         write(ipath,data,29);

        /* record size of file */
         size=_gs_size(path);
         write(ipath,&size,sizeof(size));

        /* copy the file to the ifile */
        while(size>0)
        {
                    if (size>BUFFSIZE)
numbytes=BUFFSIZE;
               else numbytes=size;
               read(path,data,numbytes);
               write(ipath,data,numbytes);
               size-=numbytes;
          }

          close(path);
    }

    /* record end-of-ifile marker */
                    for(t=0;t<29;t++)
write(ipath,"\x00",1);
     free(data);
     close(ipath);
    printf("i-file '%s' created.\n",argv[1]);
}
```

Listing #2: install_ifile.c

```
=========================
#include <stdio.h>
#include <modes.h>
#include <errno.h>

#define BUFFSIZE 8192

main(argc,argv)
int argc;
char *argv[];
{
      int ipath,path,size,numbytes,t;
     char *data,filepath[255];

    if (argc!=3)
    {
          fprintf(stderr,"install_ifile <ifile>
<directory>\n");
```

```
IF shft=2 THEN

PRINT "Right shift"
ENDIF
IF scrlck=255 THEN
PRINT "Scroll lock engaged - press again"
ENDIF
IF ctrl=255 THEN
PRINT "Control key"
ENDIF
IF alt=255 THEN
PRINT "Alternate key"
ENDIF
IF capslck=255 THEN
PRINT "Caps lock engaged - press again"
ENDIF
IF numlck=255 THEN
PRINT "Numlock engaged - press again"
ENDIF
RETURN
```

Listing #4: keydata.bas

```
=======================
PROCEDURE keydata
(* detects special keys
(* July 6, 1994
PARAM shft,scrlck,ctrl,alt,capslck,numlck:
BYTE
TYPE registers=d(8),a(8),pc:INTEGER
DIM regs:registers

DIM callcode:INTEGER
DIM  pointer:INTEGER

DIM  wdata:STRING[10]
wdata="WData"

(* first locate wdata in memory
(* with link system call
(* module pointer returned in a(3)
callcode=$00
regs.d(1)=0
regs.a(1)=ADDR(wdata)
RUN syscall(callcode,regs)
pointer=regs.a(3)

shft=PEEK(pointer+182)
scrlck=PEEK(pointer+183)
ctrl=PEEK(pointer+184)
alt=PEEK(pointer+185)
capslck=PEEK(pointer+186)
numlck=PEEK(pointer+187)
end
```

< 268'm >

Any comments, questions, or source code to be included in Joel's column may be sent in care of 68'Micros or directly to Joel at:

Joel Mathew Hegberg
936 N. 12th Street
Dekalb, IL 60115

E-mail : joelhegberg@delphi.com

Is Your CoCo Psychic?

True random number generation in DECB

(reprinted with permission from Jul/Aug 94 CoCo-Link)

Have you ever asked yourself "how random is the RND command?" Type in the following few lines and then save them.

```
10 FOR L=1 TO 10
20 PRINT RND(10)
30 NEXT L
40 END
```

Now RESET the CoCo (in this article RESET means to turn off then back on a CoCo 1 or 2, CTRL-ALT-Reset on a CoCo 3. If desired results don't occur, turn it off/on also) and run the program. Write down the numbers. RESET the CoCo and run the program again. Notice something about the second set of numbers? Try it as many times as you want. Need more proof? Change RND(10) in line 20 to RND(L), save it, and run again.

Need even more proof? Type the following short program, save it, then run a few times, RESETtin between each run. Satisfied!

```
10 X=RND(100)
20 Y=RND(5)
30 FOR D=1 TO X STEP Y
40 Z=RND(D)
50 NEXT D
```

The RND command looks at memory location 280 (&H118) for its starting point. There are two ways to get a more random RND.

First, try the following line:

```
POKE 280, PEEK(275)
```

Location 275 (&H113) has a constantly changing value. Therefore when you enter the POKE decides the starting point for RND. Load the first test program above but change line 10 to read:

```
10 POKE 280, PEEK(275)
```

Save then run the program a number of times, resetting the computer between each run and note the changing values.

Another way is the RND (-TIMER) command. On its own this command will give a long unuseable decimal number. Load the first test program, this time changing only line 30 to read:

```
30 PRINT INT(RND(-TIMER)*10+.5)
```

Save then run the program a number of times, resetting between each run. Notice the difference? Change the 10 in line 30 to the highest number needed. The "+.5" is there to compensate for the rounding down of the INT command.

Just to show that the RND(-TIMER) command isn't perfect, type in the following if you have a CoCo 3:

```
10 ON BRK GOTO 80
20 HSCREEN2
30 X=INT(RND(-TIMER)*320+.5)
40 Y=INT(RND(-TIMER)*192+.5)
50 X=INT(RND(-TIMER)*15+.5)
60 HSET (X,Y,C)
70 GOTO 30
80 END
```

For a CoCo 1 or 2, change the following lines to:

```
20 PMODE3:SCREEN1,0:PCLS
30 X=INT(RND(-TIMER)*256+5)
60 SET (X,Y,C)
```

Leave out lines 10 and 80. Run the program (no need to reset this time). Presto! Lines! Random?? Experiment with the program and modify as you wish.

Back to the original question. When you ran the first program you should get 5-4-2-7-5-4-6-3-2-9. Can your CoCo read my CoCo's CPU???

Well, there you have it— a RND that actually looks like a random number. Just as an afterthought, what if you used the RND function to select your lotto numbers? I wonder if the results would be as random as they actually seem to be? Interesting...

< 268'm >

```
        exit(0);
    }

    data=(char *)malloc(BUFFSIZE);
    if (data==(char *)NULL)
    {
        fprintf(stderr,"install_ifile: Can't
allocate buffer memory!\n")
;

        exit(ermo);
    }

    ipath=open(argv[1],S_IREAD);
    if (ipath==-1)
    {
        free(data);
        fprintf(stderr,"install_ifile: Can't
open file %s for reading.\n
",argv[1]);
        exit(ermo);
    }
    read(ipath,data,5);
     if (strncmp(data,"IFILE",5))
    {
        close(ipath);
        free(data);
        fprintf(stderr,"install_ifile: %s is
not an ifile!\n",argv[1]);
        exit(0);
    }

        while(1)
    {
        read(ipath,data,29);
        if (data[0]==0x00) break;
                printf("Installing file
%s...\n",data);
        fflush(stdout);

        /* Assemble filepath */
                sprintf(filepath,"%s/
%s",argv[2],data);
        path=creat(filepath,S_IWRITE);
        if (path==-1)
        {
            fprintf(stderr,"Error open-
ing file %s for writing.\n",fi
lepath);
            free(data);
            close(ipath);
            exit(ermo);
        }

        read(ipath,&size,sizeof(size));

        /* copy file from the ifile */
        while(size>0)
        {
                if (size>BUFFSIZE)
numbytes=BUFFSIZE;
            else numbytes=size;
            read(ipath,data,numbytes);
                write(path,data,numbytes);
            size-=numbytes;
        }
        close(path);
    }

    close(ipath);
    free(data);
     printf("Installation complete.\n");
}
```

Back in the June 1994 issue, I gave some C source code for determining if any special keys were being pressed on the MM/1's keyboard using K-Windows (CONTROL, SHIFT, ALT, etc.). Ted Jaeger has been kind enough to send us some BASIC source code to do the same thing. My thanks to him for letting me include it here! Ted also has sent me some BASIC code to show how to use K-Windows graphical buttons, which I'll be including in the next issue. As always, everyone is welcome to send in any code that they may feel will benefit other OS-9 programmers.

Listing #3 (mainloop) is a simple program which calls "keydata" (listing #4) and displays any special keys being pressed. Examine listing #4 (keydata) to learn where the keyboard data is found.

Listing #3: mainloop.bas

```
==========================
PROCEDURE mainloop
(* demo loop for checking keys that don't
alter the
(* keyboard buffer
(* July 6, 1994

DIM shft,scrlck,ctrl,alt,capslck, numlck:
BYTE

PRINT CHR$(12)
PRINT"Pressing some keys does not change
the status of the keyboard"
PRINT "buffer. These special keys include
the shift, scroll lock, control,"
PRINT "alternate, caps lock, and num lock
keys. Here is how they can"
PRINT "be detected in BASIC."
PRINT
PRINT "Press one of the special keys - I'll
report key status!"
PRINT "Press any regular key to exit."
PRINT

(* enter main loop
LOOP

(* let's exit if anything other than
(* a special key is pressed
EXITIF INKEY(#0)<>0 THEN
ENDEXIT

(* check status of special keys
RUN keydata(shft,scrlck,ctrl,alt,capslck,
numlck)

(* special key pressed
(* go report which
IF shft<>0 OR scrlck<>0 OR ctrl<>0 OR
alt<>0 OR capslck<>0 OR numlck<>0
THEN
GOSUB 10
ENDIF

ENDLOOP

END

10 (* determine which key pressed
IF shft=1 THEN
PRINT "Left shift"
ENDIF
```

Programming in "C"

The big SWITCH and Parameter Passing

P.J. Ponzo

You may recall, from an earlier lesson, that we checked various cases by using the if-else if construction:

```
1       if ( such-and-such)   {
2       ——do this——
3       }
4       else if (this-or-that) {
5       ——do this——
6       }
7       else if ( whatever )   {
8       ——do this——
9       }
10      else {
11      ——do this——
12      }
13    ——continuation of program
```

Realize that only one of the statements 2, 5, 8, 11 will be executed, depending upon which of the conditions 1, 4, 7, 10 is satisfied first. (If none of 1,4 or 7 are satisfied, then 10 IS satisfied and 11 will be executed). Even if the conditions 1 and 4 and 7 are all satisfied, only statement(s) 2 will be executed, then the program will continue with statement 13, etc. (There's a MORAL here. To speed up execution, put the most probable condition first ... then the program won't have to do so much checking). But there's another (more natural) way of checking a number of cases in C.

```
switch (x) {    /* Begin the SWITCH */
                /* on the integer x.  */
case 1:     /* If x is the integer 1, then */
   do this;  /*execute this statement*/
   and this;  /*  and this too.          */
case 2:     /* If x is the integer 2, then*/
   do this;  /*  execute this statement*/
   and this;  /*  and this too.          */
case 3:     /* If x is the integer 3, then*/
   do this;  /*  execute this statement*/
   and this;  /*  and this too.          */
   and this;  /*  and this too.          */
   and this;  /*  and this too.          */
case 4:     /* If x is the integer 4, then*/
   do this;   /*  execute this statement
...      */
default: /*If x is none of  above, then */
   do this;   /*  execute this statement*/
}
```

Notice the opening and closing brackets for the SWITCH!

```
switch (x) {
}
```

... and here's a variation:

```
1   switch (x) {    /* Begin the SWITCH*/
                    /*  on the integer x.  */
2   case 1:     /* If x is the integer 1, then*/
3   case 2:     /* If x is the integer 2, then*/
4   case 3:     /* If x is the integer 3, then*/
5     do this;   /*  execute this statement*/
6     and this;  /*  and this too.          */
7     and this;  /*  and this too.          */
8     and this;  /*  and this too.          */
9   case 4:     /* If x is the integer 4, then*/
10    do this;   /*  execute this statement*/
11  default:    /* If x is none of above,then*/
12    do this;   /*  execute this statement*/
13  }
```

If the integer x is equal to 1 or 2 or 3 then statements 5 to 8 will be executed! (...so the SWITCH will NOT STOP with the first case that is satisfied, but will check ALL SUBSEQUENT CASES!) If you don't want that to happen, then you may terminate a case with a break.

```
1   switch (x) {    /* Begin the SWITCH*/
                    /*  on the integer x.  */
2   case 1:     /* If x is the integer 1, then*/
3   case 2:     /* If x is the integer 2, then*/
4   case 3:     /* If x is the integer 3, then*/
5     do this;   /*  execute this statement*/
6     and this;  /*  and this too.          */
7     and this;  /*  and this too.          */
8     and this;  /*  and this too.          */
9     break;    /*  and now BREAK OUT*/
                /*  OF THE SWITCH!  */
10  case 4:     /* If x is the integer 4, then*/
11    do this;   /*  execute this statement*/
12  default:    /* If x is none of the above,then*/
13    do this;   /*  execute this statement*/
14  }
```

Now, if x is a 1 or 2 or 3, the statements 5 to 8 will be executed and (because of the break; in line 9) we leave the SWITCH and continue beyond line 14. If, however x is a 4, then only the statement(s) for this case are executed (line 11, in this example). Only if all cases fail will the default statement(s) be executed (for example, if x is a 6 then statement 13 is executed).

More SWITCHing

You may switch on any type of variable (not just integers). For example you may have declared x to be a char, so ...

```
1   switch (x) {    /* Begin the SWITCH*/
                    /*  on the char x.    */
2  case 'A':   /* If x is the character 'A',then*/
3  case 'u':   /* If x is the character 'u',then*/
4  case '#':   /* If x is the character '#',then*/
5     do this;   /*  execute this statement*/
6     and this;  /*  and this too.          */
7     and this;  /*  and this too.          */
8     and this;  /*  and this too.          */
9     break;    /*  and now BREAK OUT*/
                /*  OF THE SWITCH!  */
```

Note that the case comparison must be consistent with the variable type.

If x is an int then you may use case 7:
If x is a char then you may use case '+':
If x is a float then you may use case -1.234:

Think of the case comparisons as being equivalent to:

if (x==7) or if (x=='+') or if (x==-1.234) etc.

... and you may leave out the default if you wish!

CALL BY VALUE and CALL BY REFERENCE

We mentioned in an earlier lesson that a function call, in which you pass certain parameters (like average(a,b)), gives to the function copies of the parameters. The function may change these copies but the "originals" won't be changed. This is CALL BY VALUE.

You may, however, WANT to have a function change the originals. In this case you must tell the function where, in memory, the "originals" live. To do this you may pass the addresses of the parameters (or ~rpointers to the parameters). This is CALL BY REFERENCE. Knowing where the "original" parameters are, in memory, a function may now modify them.

Suppose you want to exchange() the values of two floating point numbers, say x and y, by calling upon a function exchange() :

```
exchange(&x,&y);  /* call the function, give
it addresses of x,y*/
```

The exchange funtion may look like:

```
exchange(u,v)     /* this function exchanges
two "floats".      */
float *u, *v;     /* declare u and v as pointers
to "floats".    */
{       /* the opening bracket for exchange().*/
 float temp;  /* declare a temporary float.*/
  temp=*u;          /* make it equal to "what
u points to". */
 *u=*v;  /* place the contents of v into u.*/
  *v=temp;          /* place the "temp"orary
float into v.  */
   return;   /* return ... no need to return
anything!*/
}   /*  the floats have been exchanged !!*/
```

You can try it out with:

```
main()  {
   float x=1.23, y=4.56;
      /* declare and define two floats   */
    printf("\nx=%f, y=%f",x,y);
                  /* printf their values.  */
  exchange(&x,&y);
       /* call the exchange program.      */
    printf("\nx=%f, y=%f",x,y);
         /* printf their values again!     */
}          /* that's the end of main().    */
exchange(u,v)
  /* this function exchanges two "floats".*/
float *u, *v;
  /* declare u and v as pointers to "floats".*/
{    /* the opening bracket for exchange().*/
 float temp;  /* declare a temporary float.*/
  temp=*u;
  /* make it equal to "what u points to".*/
 *u=*v;  /* place the contents of v into u.*/
  *v=temp;
  /* place the "temp"orary float into v.*/
  return;
 /* return ... no need to return anything!*/
}   /*   the floats have been exchanged !!*/
```

Now exit the text editor, saving the above with the name sam.c, then compile using cc sam, then link using link sam, then execute via: sam and get:

x=1.230000, y=4.560000
x=4.560000, y=1.230000

Here's the exchange() function again:

```
exchange(u,v)
  /* this function exchanges two "floats".*/
float *u, *v;
 /* declare u and v as pointers to "floats". */
{    /* the opening bracket for exchange().*/
 float temp;  /* declare a temporary float.*/
  temp=*u;
  /* make it equal to "what u points to". */
 *u=*v; /* place the contents of v into u.*/
  *v=temp;
 /* place the  .emp"orary float into v.  */
  return;
/* return ... no need to return anything!*/
}    /* the floats have been exchanged !!*/
```

Here's another variation:

```
exchange(u,v)
  /* this function exchanges two "floats"?*/
float *u, *v;
 /* declare u and v as pointers to "floats". */
{    /* the opening bracket for exchange().*/
  float *temp;
        /* declare a temporary pointer.*/
  temp=u;
     /* make it equal to the pointer "u".*/
  u=v;
/* make "u" point to what "v" points to.*/
   v=temp;  /* make "v" point to what*/
            /* "temp" points to.*/
  return;
 /* return ... no need to return anything!*/
}
/*  the floats have been exchanged ?????  */
```

Why won't the latter function work???

```
exchange(u,v)
/* this function does NOT exchange floats!*/
float *u, *v;
/* declare u and v as pointers to "floats".  */
{   /* the opening bracket for exchange().*/
  float *temp;
        /* declare a temporary pointer.  */
  temp=u;
     /* make it equal to the pointer"u".  */
  u=v;
 /* make "u" point to what "v" points to.*/
 v=temp;       /*make "v" point to what*/
               /* "temp" points to.  */
  return;      /* return.  */
}  /*  the floats have not been exchanged!*/
```

In this variation, the pointers u and v are copies and, although this function does change these copies of the pointers, their contents do NOT change! (so the floats never do get exchanged!).

Passing FUNCTIONS to FUNCTIONS

In an earlier lesson we computed the roots of some equation x=f(x), with f(x)=2*sin(x).

```
1  double x=1.0, y, e; /* double precision!*/
2   do  {            /* start of the do-loop*/
3     y=2.0*sin(x);     /* calculate y      */
4     e=fabs(y-x);      /* calculate error  */
5      x=y;             /* change x to y    */
6  } while(e>.0000005); /* end condition*/
7   printf("x-2sin(x)=%f when x=%f",e,x);
```

Now suppose we turn this piece of code into a function, solve() which we call via:

root=solve(f,x,e);

where we pass to solve() the function f(x), and some initial guess of the root, namely x, and an error specification e. We expect solve(f,x,e) to return a root (which, naturally, we call root!). We may write solve() like so:

```
1 float solve(fcn,x,error)
                    /*returns a FLOAT!*/
2 float (*fcn)();          /* !!!!!!!!!!!!!!!!*/
3 float x, error; /* x-value & error are floats.*/
4 {
5  float y, e;      /* declares 2 floats.     */
6  do  {        /* start of the do-loop.     */
7   y=(*fcn)(x);    /* calculates y.          */
8     e=fabs(y-x);
       /* calculate absolute value of y-x.*/
9    x=y;          /* change x to y.         */
10  }  while (e>error);
          /* check error if e is too large.  */
11 return(x);  /* return x=root if e<=error.*/
12 }
```

Line 2 has the curious declaration of fcn() as a ~rfunction pointer. The (*fcn) says it's a pointer, and the () says it points to a function and the float says this fcn returns a float! Note too, in line 7, that whereas fcn is a pointer, *fcn IS the function! (The parentheses are necessary).

```
main()  {
   float f1(), f2(), f3(), solve();
              /* declare functions used.*/
    printf("\nA root of x=f1(x) is %f",
                  /* printf the root ...   */
    solve(f1,1,.00005)); /* solve x=f1(x).*/
    printf("\nA root of x=f2(x) is %f",
                  /* printf the root ...   */
    solve(f2,-1,.00005)); /* solve x=f2(x).*/
    printf("\nA root of x=f3(x) is %f",
                  /* printf the root ...   */
    solve(f3,2,.00005)); /* solve x=f3(x).*/
}
float f1(x)
float x;
{ return(2.*sin(x)); }  /* f1(x) = 2 sin(x)  */
float f2(x)
float x;
{ return(2.-x/2.); }    /* f2(x) = 2-x/2   */
float f3(x)
float x;
{ return(1.+1./x); }      /* f3(x) = 1+1/x  */
float f1(), f2(), f3(), solve();
/* declare functions used.*/
```

Here we declare all the functions we use (they all return a float).

```
   float f1(), f2(), f3(), solve();
              /* declare functions used.*/
    printf("\nA root of x=f1(x) is %f",
                  /* printf the root ...   */
     solve(f1,1,.00005));
                     /* solve x=f1(x). */
```

Here we print (after a \newline) A root of x=f1(x) is followed by the %float returned by solve(f1,1,.00005). Note that we pass the pointer f1, a starting value 1 and an error specification of .00005

```
    printf("\nA root of x=f1(x) is %f",
                  /* printf the root ...   */
     solve(f1,1,.00005));
                  /* solve x=f1(x).       */
```

... then we continue with two more functions f2(x) and f3(x), each time specifying not only the pointer to the function but also a starting value and error specification.

(continued on page 12)

Basic09 In Easy Steps

Chris Dekker

DECB, Basic09, and MS-DOS Qbasic conversions!

Editor's Note: Due to a production error, part of last issue's column was left out. We continue in this issue with the MS-DOS BASIC (QBasic, GW-Basic, and Basic A) WINDOW command. The following starts with the last paragraph in vol. 2 no. 1 (Aug 94):

The MS-DOS Basic WINDOW statement is actually a scaling function. It defines a rectangle that is scaled up or down by basic to fill the entire screen. For instance, you have a program that only outputs values between 0 and 1 for both X and Y (a certain point being defined as X,Y) and you use the following commands: SCREEN 12 : WINDOW (0,0)-(1,1). If the program now generates the values 0,0 a point at 0,0 will be set; however if the program generates the values 1,1 this will be translated into screen coordinates 639,479. In the same way the values 0.5,0.5 will be translated into 320,240 etc.

As I said neither Basic09 nor DECB have this function built-in, so we will have to do the scaling ourselves. This isn't hard: you can accomplish it with a one line formula; you just have to get your values right.

In the Qbasic listing you see the command WINDOW (-2.4,-1.2)(2.4,2.4). We can assume that the values the calculations generate are within that range. First we figure out the scaling factor for the width of the screen: we have 640 pixels to be divided over a range of -2.4+2.4=4.8 and to center the image we must add 320 (we can't plot negative values). With X1 the value generated by the program the formula becomes: X3=320+X1*640/4.8

You can make the formula for the Y coordinates in the same way. Note that I used an offset of 116 instead of 96 (which is the screen's center line) to make sure the image wouldn't extend off the top of the screen. This is because the center line of the window's height is 0.6 and NOT 0 so the entire image has to slide down and 0.6*192 is (rounded to) 116.

I should note that sometimes you will have to adjust the offsets by trial and error to center an image on the screen, but this is a secondary consideration since you can make these adjustments after you have run the program. The same is true for the scaling factor of the circle's radius (130) which I found by trial and error. Just keep adjusting the value until the images (approximately) match.

One thing you do have to keep in mind with Basic09 (for this type of programs) is your variable typing. You must do all the calculations with REAL type variables (Basic09's default variable type) to get the required accuracy. INTEGER type divisions discard the remainders with every calculation and won't generate much of an image.

The problem with real variables is that you can not use them in calls to Gfx2. This module only accepts integers. If you make a mistake you will see an error 56 show up. There for I used the line RI=130*R instead of passing 130*R to Gfx2.

If you are good at math (or would like to think so) you probably noticed that in the formulas for calculating y3 and y4 I used a - instead of a + operator. This is necessary because Qbasic addresses the screen differently then Basic09 or DECB. While the latter use the top left corner as 0,0; Qbasic uses the lower left corner for that purpose so we will have to mirror the image along a horizontal center line to get the same picture.

And for the really sharp minded among you: we first defined our screen as 320 pixels wide and then proceeded by using the number 640 in the formulas for x3 and x4. Why?

Well,, if OS-9 sets up a window it will by default switch on a feature called scaling. This means that all windows are scaled on a 640 pixel grid. So if you want to access column 320 on a 320 pixel screen you must address it with x-coordinate 640; while you access the center column of the screen as 320.

Too difficult to comprehend? You can always insert the following line after you have established the window: run gfx2 (path,"scalesw","off"). Now all you have to do is adjust the formulas that calculate x3 and x4 and the program will run just fine.

Alternately you can change the window from type 8 to type 7 (which is basically what I did in the DECB program by using HSCREEN 4) but then you will have only four colors available.

I hope that this article has given you some insight in how windows work under OS-9/ Basic09. I have just dealt with the very basics of it all. Just how deep you will have to dig into the windowing system (and the problems that presents) depends on the needs of the program you are working on.

If you have any questions about these (or related) matters please feel free to contact me or this magazine's editor. Next time I will look at converting some specific DECB functions and statements to Basic09.

Next Time!

As promised in the last installment, this time we will take a look at converting commands from DECB to Basic09. First of all there are a number of commands in DECB that are NOT supported under Basic09 so if your program contains any of those you can start stripping them right away.

For starters there are the commands dealing with the cassette port. The way in which the CoCo reads the cassette port demands constant attention from the microprocessor during data transfers. Because OS-9 is a multi-tasking operating system it can not pay that kind of attention to a particular application. (even when you have only one program running OS-9 will interrupt it 60x per second to do it's own chores.) So out go CLOAD, CLOADM, CSAVE, CSAVEM, SKIPF, AUDIO and MOTOR. If you have to you can still control the motor relay from within Basic09: poke $FF21,52 will turn it ON, while poke $FF21,48 turns it OFF.

The next batch of commands to go are the ones dealing with memory allocation and machine language. LPEEK and LPOKE are not supported because OS-9 partitions the computer's memory and you can not directly access memory locations outside your own (64K) workspace.

DEFUSR, USR and VARPTR are not supported because under Basic09 you use a very different way of interacting with ML subroutines: the subroutine has a name and you call it with a RUN statement. EXEC is not supported either because under OS-9 code has to be position independent so there is little use in trying to execute code at a certain address because you don't know if there is any code there or just "garbage".

The DEFFN label is also not recognized by Basic09. You can still use the formula involved with the function but you will have to make it part of the calculations, rather than predefining it.

PLAY is not supported either, which is just as well: under OS-9 you pretty much need special drivers to do a good job with sound (unless you like the sound of machine guns).

WIDTH is not supported. If you have to set up a text window use the gfx2 module (this was discussed in part 4) and define the window as type 1 (40 column) or type 2 (80 column).

STRING$ is another casualty. If you want to define a long string of repeating characters under Basic09 you must either type all the characters or define a short string and add it a number of times to itself using a loop.

The last command in this row is TIMER. Basic09 does provide you with a timer (DATE$) but this variable holds the current date and time. It's smallest time increment is a second, unlike TIMER which counts 60 ticks per second. (Actually OS-9 keeps time using those same ticks but they are not available to programs.)

I know that there are more commands/ functions not supported by Basic09, however rather than throwing them all on one heap I will try to keep the story a little structured. I am currently skipping all commands that are part of the "DISK" ROM of DECB. For OS-9 a lot of these functions are part of the operating system itself and not of Basic09. This results in a lot of those functions being accessed indirectly through the syscall utility rather than with Basic09 commands.

The same story is basically true for I/O through the serial port. Screen functions (both text and graphics) are handled through the gfx2 module (gfx for lower resolution screens) and the basics of that have been discussed in the last part of this series.

Another big difference between Basic09 and DECB is that Basic09 has three different modes whereas under DECB you're always in the same (interactive) mode. What I mean is this: under DECB you can enter program code, edit it and execute it (or directly execute commands) all from the OK prompt.

Basic09 on the other hand has several different modes each with it's own possibilities and limits. RENUM for instance can be use in edit mode but not in any other. When you start Basic09 you end up in the COMMAND mode. Here you can use commands like mem, edit, run, dir, etc. (see page 10-9 for a complete list). As you can see a command like NEW is not supported but typing KILL* will have exactly the same effect. NOTE you are in the command mode whenever your prompt reads B:.

Basic09 also has an EDIT mode (page 10-10; prompt is E:) that you can enter from the command mode. In this mode you enter your program code and Basic09 also checks your program for errors. Unlike DECB you do not automatically exit this mode after a line is edited.

A third mode is the DEBUG mode (page 10-11; prompt is D:). This mode is automatically entered whenever a running program runs into an error, unless you trapped that particular error. Two other ways of entering the debug mode are inserting a PAUSE instruction in the program or pressing <CTRL><C> from the keyboard when a program is running.

Note that this mode is only for getting the final bugs out of a program. As long as the edit mode reports errors in your program you can not start it. Also, once your program is PACKed it can no longer enter the debug mode when it encounters an error. If you have not dealt with that error in the program it will simply crash. (I suppose you could say running packed programs is a fourth mode.)

Now back to conversions. Fortunately there are also a considerable number of commands that do the same job under both languages. Some of them control program flow and most mathematical functions also fall into this category. Here's a list of them: READ/DATA, END, (ON..) GOSUB, (ON..) GOTO, INPUT, LET, POKE, REM, RESTORE, RETURN, RUN, STOP. I suppose I could add DIM also to this list but you can not dynamically DIMension variables as I mentioned in part 5.

A list of functions that need no further attention: ABS, ASC, ATN, CHR$, COS, EXP, FIX, INT, LEFT$, LEN, LOG, PEEK, RIGHT$, RND, SGN, SIN, STR$, SQR, TAN, VAL. All operators (+,-,>,=,etc.) fall in this category as well.

If you want to port over an entire program without retyping it, your best bet is to make a copy of the program in an ASCII file (save with ,A option) and, after copying it to an OS-9 disk, first use OS-9's edit command or word processor to do some preprocessing.

The reason for this is fairly simple: whenever you make a change in a line using Basic09's editor, that line will immediately be checked for errors. If one is found the editor aborts the command you gave it and points out that error to you.

Normally this works great, but when porting a program this may be less desirable. For instance if you use more than one statement per line DECB uses : as a separator. Basic09 on the other hand uses \ This is no big deal if you can replace them with one global replace command. On the other hand if you have to replace them one by one it becomes very frustrating. Since OS-9's edit command acts more like a word processor and doesn't perform error checking it is more suited for the task.

And now for the hard (or frustrating) part: dealing with the commands that look oh so familiar, but behave just a little different. The first one is CLOSE. Under DECB CLOSE has a number (channel) attached to it while under Basic09 it has a variable. This variable has been defined earlier in the program (by an OPEN statement for instance). If you got that working you shouldn't have too much trouble with CLOSE. For INPUT# the same rules apply as for CLOSE.

IF .. THEN .. ELSE has the same control function under Basic09. It is just a little pickier so keep the following things in mind: ALWAYS terminate the block with an ENDIF statement because Basic09 does not see the end of a line as an implied endif. Basic09 accepts the ELSE statement only if it is the first word on a new line OR if it directly follows a backslash (\). If your IF .. THEN statement points to a line number you MUST omit the ENDIF part otherwise Basic09 generates an error.

With MID$ we are back to the old days: under Basic09 this command behaves in the same way as it did under color basic (before extended came along). It simply returns a portion of the target string rather than replacing it with a new string. The syntax, however, is exactly the same as for the DECB command.

Actually replacing a piece of a string under Basic09 is a rather tedious operation: you will have to split the original using left$ and/or right$ and then build a new string by concatenating the various pieces: A$=B$+C$+D$ etc.

A FOR..NEXT loop is much the same as in DECB, but you have to keep in mind that you MUST specify a variable in your NEXT statement otherwise Basic09 will report an error. Another thing is that Basic09 wants to see one NEXT per loop. Terminating multiple loops (e.g. NEXT I,J,K) with one statement doesn't work.

ON ERR GOTO must be changed a little bit into ON ERROR GOTO, but has the same function in both languages.

The OPEN statement has more or less the same function in Basic09 but a very different syntax. In it's generic form it looks like OPEN #path,filename:accessmode. For your program to work "path" must be a byte or integer type variable. Your program can later use that variable to access the file but it can NOT assign a value to the variable. OS-9 will do this for you and if you change it's value you will no longer be able to access the file. "Filename" can be a (string)variable or a literal string. If you use the latter option you must enclose the name in quotation marks. "Accessmode" under Basic09 is not defined with a letter but with a word. Most commonly used are READ, WRITE, and UPDATE (which allows reading and writing to a file). Two more options are DIR and EXEC. DIR allows you to access directory files, while EXEC causes Basic09 to work with the execution directory as opposed to the data directory. You may specify more than one accessmode if necessary.

PRINT behaves mostly in the same way as under DECB except the PRINT # statement which is generally used with a variable. i.e. PRINT #path. OS-9 reserves the three lowest pathnumbers (0,1,2) of every process for input, output and error paths. So if you use the statement PRINT #1,"A" you will see a capital A appear on the screen. By default the error path (#2) is also connected to the screen and the input path (#0) is connected with the keyboard. Since all characters typed are echoed to the screen, PRINT #0, PRINT #1 and PRINT #2 have the same effect until you redirect the input and/or output paths. You do so with the <, > and >> modifiers on the OS-9 commandline. In those cases your message will be printed to whatever device you choose: another window, a printer, diskfile, etc.

PRINT @ is not supported by Basic09. For positioning text on the screen see the description of LOCATE.

PRINT USING under Basic09 is much more comprehensive than with DECB. Starting at page 11-122 Basic09's manual devotes 7 pages to explaining it and I don't feel like repeating that over here. A few quick points will do: Basic09 recognizes 6 formats: string, boolean and 4 number types. For most formats (including strings) you can use the <, > and ^ modifiers to get things in place. Unlike DECB you do not repeat a character but use a number-symbol combination to tell Basic09 the size of the field it can use. For instance ##### would become I5 if the value is an integer or R5.0 if it is a real number, while % % would be replaced by S6.

Note that Basic09 also recognizes Txx for tabulating and Xxx for inserting spaces. The entire string of symbols must be enclosed in quotation marks, followed by the various variables that have to be printed out.

For instance PRINT USING "x2, s10<, R7.2^","Total",amount would be printed in this fashion:

Total 100.00 if amount=100.

If you use the tabulation functions [TAB(xx) or Txx] keep in mind that they behave differently if output is send to the printer as compared to the screen. For instance the line PRINT #path, TAB(25);filename will print a filename starting at column 25 when path points to the screen. This will always work as long as the current cursor position is somewhere in columns 0-24. However if path is connected with your printer the same line always results in printing 25 spaces, regardless of the position of the printhead.

GET and PUT under DECB are graphics functions. Although they have the same functions under Basic09 when used in conjunction with the Gfx2 module [i.e. RUN Gfx2("get",...)] they also have a very different function in the form of the Basic09 GET and PUT statements. In that case they are simple I/O functions like INPUT and PRINT.

There is one big difference between GET/PUT and INPUT/PRINT: while the latter two will format the data they deal with, GET/PUT will just transfer it. For instance if you type DIM buffer(500):BYTE and then GET #path, buffer GET will (attempt to) read 500 bytes via that path. Suppose that path is associated with a file. GET now reads the first 500 bytes after the filepointer regardless of the meaning of the data. It could be 250 integers, 100 real numbers or 10 50-character strings; that just makes no difference at all. PUT will perform the same way but writes data to a path. These commands are mostly used with complex datastructures where you can mix various types of variables in one record. With GET/PUT you can read/write the entire record (or array for that matter) with one command. Be careful when using GET for input from the keyboard. For example if your program has the instruction GET #0, key but does not define "key"; your computer will "hang up" until you have pressed 5 keys. The reason for this is that Basic09 regards every undefined variable as a REAL number which is made up of 5 bytes in memory. So the instruction insists on reading 5 bytes before returning control to Basic09.

INKEY$ is not directly supported by Basic09, but shipped as a separate module. Consequently a statement like A$=inkey$ translates into RUN INKEY(A$). Make sure the inkey module is available to OS-9. It is generally a good idea to merge it with Basic09 and/or the Runb module into one file.

LOCATE is not handled by Basic09 either but by the Gfx2 module and more specifically it's CURXY function. LOCATE 20,12 translates into RUN Gfx2("curxy",20,12).

INSTR has an equivalent under Basic09 that is called SUBSTR. There is one big difference between the two: with SUBSTR you can NOT specify a starting column. If your program is absolutely dependent on that your can translate A=INSTR(5,search$,"find") into the following code:

```
temp=right$(search$,len(search$)-5)
A=substr("find",temp)
```

Note that with SUBSTR search- and target string are exchanged compared to INSTR. SUBSTR will return 0 if it can not find a match in temp.

EOF returns TRUE or FALSE under both languages, but it's syntax is slightly different under Basic09. Typically one uses the statement IF EOF(#path) THEN with Basic09 to determine whether the end of a file has been reached. Note that in this case "path" refers to the variable used in the OPEN statement.

The JOYSTK function is not supported under Basic09. That doesn't mean you can not read the joystick ports, it just means it takes a little extra work to do it. Under OS-9 we have to read them through a system call. First you must set up a data structure representing the 6809's registers. How to do this was shown in an earlier part of this series.

To execute this particular call use the following code:

```
regs.a=1\regs.b=$13
regs.x=0 (left stick; use 1 for right)
run syscall($8D,regs)
```

Register A returns the button status (a code ranging from 0-3); while regs.x returns the x coordinate and regs.y returns the y coordinate of the joystick's position. For more info on this call see page 8-113 of the technical reference section of your OS-9 manual.

ERROR TRAPPING. Error trapping is basically the same thing under both languages, but the various functions have different names. I already discussed ON ERROR GOTO. ERLIN is not supported because, by default, Basic09 uses no line numbers. The ERNO function is called ERR under Basic09. Your basic error trapping routine would look something like this:

```
ON ERROR GOTO 100
100 errnum=ERR
```

Note that we must catch the value held in ERR in a variable because Basic09 resets it's value to 0 as soon as you read it. If an error occurs in this example ERR will hold the errorcode, while program execution jumps to line 100 and continues from there.

The ON BRK GOTO statement is not supported under Basic09 either. However since pressing the BREAK key will invariably get you an error 2; you can replace this statement by IF errnum=2 THEN sometime after line 100 (in the error trapping routine). Note that you can not test this feature if you run your programs from within Basic09 because Basic09 traps this error and interrupts the program rather than passing on the error code to it. However once you have PACKed your program and run it using the Runb module it will work.

The last thing I want to mention is that Basic09 supports an extra command called ERROR. Whenever Basic09 encounters this command it generates an error and then jumps to the error trapping routine to deal with the error. This comes in handy for debugging a program or when your program has "dead ends" in it. You can terminate that piece of code with an ERROR statement followed by a number and deal with the situation in your error trapping routine.

Basic09 has a lot more commands that are not supported by DECB but since you don't really need them when converting a DECB program I won't deal with them here. Next time we'll take a look at the commands inside the "DISK" ROM of DECB.

Chris Dekker
RR #4
Centreville, NB E0J 1H0
CANADA

< 268'm>

MM/1 Update

David Graham

BlackHawk and the MM/1 in Chicago!

Well, here we are, early August as I write, and MM/1 I/O boards and 8 meg backplanes are on order - and I should have the bare boards in stock by the end of the month. We expect to ship by the middle or end of October, and are now accepting orders for 8 meg bus cards and complete MM/1 systems. BGFX, Serial Paddle boards, and MIDI ports are in stock and ready to ship now, and of course, the 68340 upgrade cards remain available, shipping within 3 weeks of receipt of each order. MM/1 Tech manuals will be available by the time you read this. I have received the master copies, but have to work out printing and hole punching services at a price low enough to match the markets demanded retail price.

A new item is StrongWare's Copy Cat. Now in stock, you simply have to see this multiple configuration Simon(tm) type game! John and his Team OS-9 have done a marvelous job. We'll soon be beta testing his paint program, name to be announced. The hires support offered by this powerful package will make it a MUST have for the MM/1 artist!

On the communications front, Bob Billson's UUCP package is being bug-fixed even as we speak. Even with the minor problems reported from the field, this has been one of the best, and most complete implementations of UUCP ever on the OS-9 platform. Look for some impressive developments here. John Donaldson is working on a port of RiBBS and Binkly. These packages should allow OS-9 users to select a newer and more powerful platform to run their Fido Net nodes on, while remaining with OS-9. I expect that the first release will be a PD/Shareware version called SparRiBBS (Spare Ribs is indeed how this is pronounced, and I claim credit for perpetrating this abomination upon the OS-9 community - devious, aren't I??). And KTerm version 2.01 is now available.

From Arkansas, the MM/1 support team is joined by Ray Patterson. Ray's experience with OS-9 started with OS-9/6809 version 1.1's release, so he goes back quite a ways! An experienced teacher of programming at the college level, an electronics technician, and a veteran of CD-ROM work at the University of Arkansas, Ray will be active in supporting educational development on K-Windows machines. His first software projects will involve development of a complete file manager and driver set for CD-ROM and CD-I formats. Ray will also be taking up hardware support for the MM/1 , as soon as we get him copies of the schematics, etc. He has already been helpful in handling our needs for PAL programming for new MM/1 production. Thanks Ray, and welcome aboard!! Ray can be reached at Patterson Electronics, PO Box 386, Mountain View, AR 72560.

I am told that Kevin Pease has shipped the source to K-Windows to Carl Kreider and the documentation files to me. So look for full documentation for KWindows sometime in the next 6 months. There will be a charge for these packages, to cover expenses, though it won't be too much. My documentations specialist is currently revising the BGFX documentation, and the MM/1 Users Guide, as well as development of documentation for the 8 megabyte DRAM upgrade cards, so it may take awhile longer. Kevin Darling has promised me that Atari K-Windows will be made available to me for retail release as soon as he can get to it. Unfortunately, his employer ships him all over the US and Canada, so it may be a while. As you can see, we remain quite active, though at this point, we have many irons in the fire, and each step usually requires my personal involvement.

Comments are always welcome, and we can now be reached on FidoNet's OS-9 Echo, CoCo Echo, MM1_Tech Echo, or directly at Molly's Playhouse BBS (405) 233-3866, 24 hours, 14.4K baud. Look for the OS-9 Chatter echo. Hang in there, and keep on OS-9'in!

PS - Just a quick note before we go to press! I have bare I/O and 8 meg cards in hand, and arrangements to stuff them are proceeding. Ray Patterson has contacted Mark Griffith, and has his full stock of spare parts, schematics and MM/1 boards in various stages of repair, minus any boards that MM/1 owners had sent in for Mark to alter. Ray expects to be able to turn boards around more quickly, as he has a rather complete electronics workbench. (Mark's trouble shooting activities were rather limited by his lack of an oscilloscope for example).

I'd like to take this opportunity to thank Mark for trying to fill a gap like this for so long and helping Ray get set up for a smoother transition. This will allow us to concentrate on seeing that things go right in the future, without dwelling on the errors of the past. See you in Atlanta!

< 268'm>

Micro News

I have finally found an excellent book that should help any OS-9 user (Level 2 or 68K). It's called Practical C programming by Steve Oualline. It's published by O'Reilly & associates Inc. (The best UNIX books ever made!). This book covers UNIX and DOS C programming with most emphasis on C under UNIX. This resembles the OS-9 world much more so than any DOS books on the subject. It is written for new comers to C. It covers style and how to use the make utility and other good tidbits besides just code slinging. Advances through arrays pointers structures serial I/O and bit wise operations. At 396 pages, this book should work nicely for any OS-9 user who wants to learn C. The cost was $29.95. Any bookstore should be able to order it for you. (from Mike Rowen)

Corrections for Patch OS-9 v1.1
There were a couple of errors recently discovered in the Patch_OS9.src source code. It is very easy to correct using either the Basic09 editor, OS-9's Edit, or any other editor. After Patch_OS9.src has been corrected, load Basic09 and PACK the file according to the manual. Then delete the file "Patch_OS9" from the CMDS directory on disk.
If you have already run Patch_OS9, there is no need to run it all over again. The error refers to the Kevin Darling SCF patches. Simply look up the instructions in the DOC directory for manually installing the SCF patches. These patches allow editing the command line by moving the cursor with the arrow keys rather than CTRL-A and patch a "disappearing window" bug. Both should be installed (or neither).
There is also a minor problem with the standard bootlist. WindInt.io in line 11 should be changed to grfint.io. The MultiVue bootlist (bootlist.mv) should be correct as is.

To correct Patch-OS9.src for auto installation:
1) After loading the file in an editor, find the lines that read:
PRINT"Patch SCF- Full Editing"
PRINT"Patch SCF- Disappearing Window Bug"
2) Delete the line immediately following the above. (SHELL "ipatch /d1/ipc/scf.ipc /d1/modules/scf.mn_Tandy /d1/modules/scf.mn")
3) Add the two lines below in place of the deleted line:
SHELL"ipatch/d1/ipc/scfkd.ipc /d1/modules/scf.mn_Tandy /d1/modules/scf.mna"
SHELL "ipatch /d1/ipc/scfkd.wp.ipc /d1/modules/scf.mna /d1/modules/scf.mn"
4) Reverse the positions of the next two lines, adding a space between the second added line and the next line.
SHELL " ipatch /d1/ipc/scfkd.wp.ipc .....
ENDIF
SHELL "del /d1/modules.scf/mna"
5) Save the edited file. Load into Basic09 and PACK. Delete Patch_OS9 in the CMDS directory and replace it with the new PACKed file.
(re)Name the new file Patch_OS9.
6) If you run the new file, there are a couple more steps necessary to utilize the SCF patches.
6a) For each window (including TERM, if used) type: xmode /(window)-bsb reprint=9 dup=19
6b) When finished, type: del/dx/os9boot; cobbler/dx (/dx is the drive with the boot disk in it)
This will replace the boot file with a new one. The new boot will automatically initialize all the devices with full editing capabilities on startup. Reboot system to check. If a new window is added later, repeat the above for the new device.
I'm sorry for any inconvenience this may have caused. If you have version 1.2 (after July 94) the above corrections have already been made. Version 2.0 with the latest shell plus should be out shortly (shell+ 2.2 is also on this issue's "microdisk").

New OS-9 software has been sighted from Chris Dekker and Terry Simons! Some time ago Terry Simons (editor of Mid Iowa & Country CoCo Club's UPGRADE disk magazine) wrote a DECB program that kept home and small business financial records (check book register, credit card balances, savings account, etc.). To this he also added a mailing list database capable of printing labels, a simple form maker utility, and a disk jacket directory printer (to be taped to a disk jacket). This integrated package was called "Home-Pac". Recently Terry gave Chris Dekker the basic source for Home Pac and allowed it to be ported to Level II. Chris has done a great job, and this versatile tool is now available for OS-9 Level II from Chris Dekker. If you need a good check book program that does a little more than just checks, "check" this one out!

This neat little utility by Joe Carson deletes the directory on a DECB disk. If a disk was formatted, you can now delete the directory only and reuse the disk... sort of a "quick format" utility!

```
0 REM CLEAR-DIR VR 1.00.00
APRIL94
1 REM BY JOE CARSON
2 CLEAR 256
3 POKE&HFFD8,0
4 INPUT"DRIVE #";DN
5 A$=STRING$(128,255):
6 FORA=1 TO 18:DSKI$DN,
17, A,A$,A$:NEXT A
7 UNLOAD
8 END
```

Can anyone whip up a similar "quick format" for OS-9?

I used to be a big Infocom game fan when I had my CoCo and used DECB (long ago). When I moved to OS-9, I didn't play them anymore since doing so meant going back to DECB.
While fiddling around on chestnut, I ran across an Infocom game interpretter that Brian White had ported to run under OSK. After downloading it, I found that Infocom sells "Lost Treasures of Infocom" Vols. I and II, a compendium of their original games. After calling around a few local software shops, I located a place which was selling both volumes. Although I just picked up Vol II, there are a ton of games on both:

Vol I: Deadline, Suspect, MoonMist, Spellbreaker, Ballyhoo, Infidel, Hitchhiker's guide to the Galaxy, Starcross, the Witness, Planetfall, Suspended, Stationfall, Enchanter, The Lurking Horror, Zork I, Zork II, Zork III, Beyond Zork, Zork Zero, Sorceror, Suspended

Vol II: A Mind Forever Voyaging, Bureaucracy, Cutthroats, Hollywood Hijinx, Plundered Hearts, Nord and Burt, Sherlock Holmes, Titan, Trinity, Wishbringer, Boarderzone, Seastalker

The store that carried them said that they are no longer a hot item. Vol I was reduced from $69.96 to $39.95. Vol II was reduced from $39.95 to $19.95.
All that was necessary was to insert the 3.5" MS-DOS disk in my MM/1, use pcf to copy the .DAT files from each of the four disks, and voila! I now have 12 games sitting in my GAMES/INFOCOM directory which can be played at my whim. The interpretter works flawlessly! I now have Infocom games on my MM/1, and it looks really cool. (from Boisy Pitre)

If you have new soft or hardware products, let us know! We will gladly print a free blurb for you here in MicroNews whether you advertise or not (though we will be happy to have your ad also).

ITEMS FOR SALE:
New in box CoCo 3 and FD-502 disk drive w/ controller, plus used CM-8 RGB monitor- asking **$300 for all.**
Also have numerous CoCo hardware and software for sale, call for list.
Bill Sgambati 914-356-6553
(after 6pm EST)

NEW PRODUCTS
from
FARNA Systems!

CoCo Family Recorder/OS-9 1.0

If you are into genealogy, then the CoCo Family Recorder is the *absolute best* program for the CoCo 3. But it runs under DECB, and many OS-9 users simply don't want to leave the multi-tasking environment. Well, we now have a solution for you! CCFR has been ported to OS-9! The program is very easy to use and menu driven. The OS-9 version is nearly identical to the DECB version in appearance, but takes advantage of many OS-9 features such as pop-up windows for entering data. DECB users can send their *original* CCFR disk (it will be returned) to get the OS-9 version for only $20.00. Others must pay the regular price of **$28.50**. Requires at least one 40 track double sided drive (FD-502) or larger. Can be shipped on 3.5" 720K disk if requested.

FARNA Systems
Box 321
Warner Robins, GA 31099
$2.50 shipping and handling per order. Canada S&H $4.00; Overseas $7.00

OS-9 Point Of Sale 1.0

If you have a small retail store business, this may be just what you've been waiting for! Designed specifically for the small business that needs more than one check-out station but can't afford the $7,500.00 or more for an MS-DOS based system. This easy to use, menu driven software uses OS-9's inherent multi-user/tasking features, eliminating the high cost of DOS based networks. Has all necessary features to replace your cash register and keep track of your sales and inventory automatically. Supports multiple serial ASCII terminals.Current price is only **$62.50**.

FARNA Systems

Box 321
Warner Robins, GA 31099
Phone 912-328-7859
Internet: dsrtfox@ delphi.com

Software, Books, and Hardware for all OS-9/OSK Systems!

CoCo DECB Software:

CoCo Family Recorder - $17.50
Genealogy program for CoCo 3. Requires 2 drives, 80 col. monitor.
CoCo Family Recorder /OS-9 - $28.50
Same as DECB version for Level II. Comes with utility to convrt DECB files. Upgrade for $20 by sending DECB disk (will be returned)

DigiTech Pro - $12.50
Record any sound for easy play-back in your BASIC or M/L programs. 512K required.

ADOS: Support for double sided drives, 40/80 tracks, faster formatting, much more!
Original (CoCo 1/2) - $15.00
ADOS 3 (CoCo 3) - $25.00
Extended ADOS 3 - $30.00 (ADOS 3 req., RAM drives, support for 512K-2MB)
ADOS 3/Ext. Combo - $50.00

DECB Games:
Mind Games - $7.50
Collection of 9 classic games.
Cross Road II - $7.50
Tic-Tac-Toe with sound and graphics!
Space Intruders - $14.50
Looks like "Space Invaders"! CoCo 1/2 and 3.
Donut Dilemma - $14.50
Climb, jump, and ride elevators to top of Donut factory to shut it down! 10 levels. CC 1,2,3.
Rupert Rythym - $14.50
Great action adventure!
Get Space Intruders, Donut Dilemma, and Rupert Rythym for only $33.50! Save $10!

CoCo OS-9 Software:

Patch OS-9 - $7.50
Automated program installs most popular/needed patches for OS-9 Level II. 512K and two 40T/DS (or larger) drives required. (*128K/35T users can install manually- state 35T.*)

OS-9 Point of Sale - $62.50
Maintain inventory, print invoices, customer catalog, etc. Multi-user capable under Level II. Supports ASCII terminals. Basic09 required. Simple menu driven interface.

***NEW!* Pixel Blaster 1.3 - $20.00**
Easy to use graphics/animation creating utilities for Level II.

Books:

Tandy's Little Wonder - $22.50
140 page softbound book with history and technical info for all CoCo models. Schematics, peripherals, upgrades, modifications, repairs, much more- all described in detail!

OS-9 Quick Reference Guides
Level II (Revision 2) - $7.50
68K (based on 2.3) - $10.50
These handy QRGs have command syntax, error codes, special keys functions, etc., in a 5.5"x 8.5" desk-top size.

CoCo Hardware:

DigiScan Video Digitizer - $150
Capture images from VCR, camcorder, or TV camera. Uses joystick ports. CoCo Max3, Max 10, Color Max 3 compatible. Special order- allow 90 days for delivery. Send $75 deposit.

Ken-Ton SCSI Hard Drive System and Components

Complete, ready to run, "plug and play" 85MB system. Top quality drive, case, and ps. Send how much space for DECB, OS-9 --- **$550.00**
No-Drive Kit: controller, OS-9 drivers, RGB-DOS in ROM, 2 pos. "Y" cable, and drive cable (specify type). Seagate N series drive with ROM rev. 104 or greater needed. ---- **$250.00**
(state how much of drive to be used for OS-9)

Controller only ---------- $135.00
OS-9 Drivers ---------- $25.00
RGB-DOS (for DECB access)---------- $35.00
** $50 for RGB-DOS and OS-9 drivers when purchased together with a controller **
"Y" cable, $25 for two position, $35 for three.
Drive cables - specify direct to drive or SCSI case type connector ---------- $25.00

Add $2.50 S&H per order. Canada/Mexico add $4.00; Overseas $7.00

FARNA Systems Publishing Services

Type Setting and Printing: We can prepare professional typeset manuals, books, booklets, catalogs, and sales flyers for you - we can print or you reproduce as needed from a master set! Very reasonable prices - inquire!

Mailing Service: If you send catalogs or letter correspondence to 200 or more persons at once, we can do all work for you for about the same cost of your materials alone! How much is your time worth???

Contact Frank Swygert at above address/phone for quotes

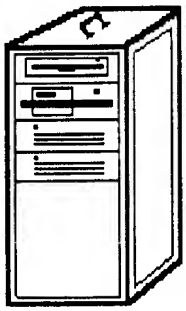
for all your CoCo hardware needs, connect with

CoNect

449 South 90th Street
Milwaukee, WI 53214
E-mail: rickuland@delphi.com

NEW PRODUCT: The Cube

This tower enclosure was designed specifically for the CoCo and peripherals, even an MPI if desired! Four drive bays, two will hold a pair of 3.5" drives sideways. Easy access, carry handle mounted on top!



Mini RS-232 Port: Don't let the name fool you! This is a full featured serial port, supporting the signals needed for flow control as well as the basic 4. Jumper blocks allow readdressing or swapping DSR/DCD. No custom cables or hardware widgets needed here! Y cable users will need to add $9.95 for a power supply. **$49.95**

Hitachification: CoNect will install a Hitachi 63B09E CPU and a socket into your CoCo. Machine MUST be in working condition! The 68B09E will be returned unharmed. 90 day limited warranty. Chip and installation only **$29.95**

REPAIRS: We can repair most damaged CoCos, even those with bad traces where a 68B09 was removed. Costs vary with damage. Bad 68B09 sockets repaired for only $40! **Inquire BEFORE sending your computer.**

DISTO Products

Quality hardware for your Color Computer!

2MB Upgrade (no RAM) - $99.95
Mini Disk Controller - $70
Super Controller I - $100
Super Controller II - $130
MPROM Burner - $50
3-N-1 (parallel, RS-232, RTC) - $75
SASI/SCSI Hard Disk Adapter - $75
Full Turn of the Screw - book with all Tony DiStefano's Rainbow articles, January 1983 to July 1989 - **$20**
Complete Schematic Set - all DISTO product schematics except 2MB upgrade - **$20**

Include $2 S&H for book or schematics. Hardware S&H is $4.50 for one item, $6.50 two or more. Certified check or International Money Order only!

Running low on some items- please call for availability!

514-747-4851

DISTO

1710 DePatie,
St. Laurent, QC H4L 4A8
CANADA

By arrangement with **StrongWare**, Sub-Etha Software is proud to bring you...

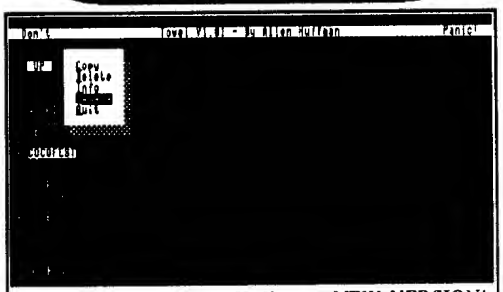
Soviet Bloc - Called the best Tetris(tm)-like game by many, this is a bigger version of the now-classic falling shapes puzzle game. RS-DOS Req: CoCo 3, Joystick/Orchestra-90 Pak optional _____ $19.95

GEMS - "Columns" of colors fall as you change the order of the colors. Match three in a row, column, or diagonal at the bottom and those colors disappear. Sounds simple, doesn't it? RS-DOS Req: CoCo 3, Joystick/Orchestra-90 Pak optional _____ $24.95

Copy Cat - Simon says "match the sequence of tones as the colored diamonds flash". Great for building memory skills. RS-DOS Req: CoCo 3 _____ $ 9.95
OSK Req: MM/1 or 100% K-Windows Systems _____ $14.95

HFE (Hprint Font Editor) - A fantastic editor for those HPRINT fonts with lots of options. Create your own character set which you can LOADM and us in your own programs. Also creates and edits fonts compatible with MiniBanners! RS-DOS Req: CoCo 3, Joystick optional _____ $19.95

Font Collection - A collection of 18 useable HPRINT or MiniBanners fonts. (Importable to OS-9 for use with MiniBanners09 as well!) RS-DOS Req: _____ $ 9.95



Towel! V1.01 by Allen Huffman - NEW VERSION! A program no intergalactic hitchhiker should be without! Use mouse or keyboard hot-keys to perform common disk and file commands from pull-down menus. Tag multiple files for Copy, Delete, Rename, etc., and even point 'n click a Backup, Cobbler, Dcheck, or whatever. User definable menu for custom options. Runs under the EthaWin interface (included) on a high-speed text screen. All commands/colors configurable. OS9 Req: CoCo 3, OS-9 Level 2 _____ $19.95

Sub Etha Software
P.O. Box 152412
Lufkin, TX 75915

Add $2.50 S&H. Texas residents add 8.25% tax. Write us for more info!

Sub-Etha Originals:

MultiBoot V1.03 by Terry Todd & Allen Huffman - Type "DOS" and bring up a list of up to sixteen OS9BOOT files! No more floppy swapping. A serious must-have for intense OS-9 users. OS9 Req: CoCo 3, OS-9 Level 2 _____ $19.95

1992 CoCoFest SIMULATOR V1.02 by Allen Huffman - NEW VERSION now uses compressed graphics and has scoring. Take a walking tour of the '92 Atlanta CoCoFest with this graphics adventure. 16-level digitized photos of the event and a text parser (ie, "get the box of disks") to let you interact. Runs on a 640x192 graphics screen. OS9 Req: 512K CoCo 3, OS-9 Level 2, 500K+ of Disk Space _____ $ 9.95
OSK Req: MM/1 or 100% K-Windows System _____ $14.95

Write-Right by Joel Mathew Hegberg - Featureful word processor for the MM/1 with what you would expect from a "real" word processor. What you see is what you get! _____ $54.95

Etha-GUI by Joel Mathew Hegberg - A neat program launcher for the MM/1 which includes handy desktop utilities like a phone dialer and nifty screen savers...even a trash can. Point and click icons to run programs. _____ $34.95

Quality OS-9 Software from

ColorSystems

NEW! K-Windows Chess for MM/1
Play chess on your MM/1.....$24.95

NEW! X-10 Master Control for MM/1
Use MM/1 to control you home!.....$29.95

Variations of Solitaire
Pyramid, Klondike,Spider,Poker and Canfield
MM/1.....$29.95 CoCo3.....$19.95

OS-9 Game Pack
Othello, Yahtzee, KnightsBridge, Minefield, and Battleship
MM/1.....$29.95 CoCo3.....$19.95

WPShell
An OS-9 Word Processing Point and Click Interface
CoCo3.....$14.95

Using AWK with OS-9
Includes V2.1.14 of GNU AWK for OS-9/68000
MM/1.....$14.95

To order send check or money order to:

Color Systems
P.O. Box 540
Castle Hayne, NC 28429
(916) 675-1706

Call or write for a free catalog! Demo disks also available.
NC Residents please add 6% sales tax
Owned and operated by ZacK C. Sessions

EDTASM6309 Version 2.02 ---------- **$35.00**
This is a major patch to Tandy's Disk EDTASM to support Hitachi 6309 codes. Supports all CoCo models. CoCo 3 version uses 80 column screen, 2MHz. YOU MUST ALREADY OWN TANDY'S DISK EDTASM TO MAKE USE OF THIS PRODUCT. It WILL NOT work with a disk patched cartridge EDTASM.

CC3FAX ---------- **$35.00**
Extensive modification to WEFAX (Rainbow, 1985) for 512K CoCo 3. Uses hi-res graphics, holds full 15 min. weather fax image in memory. Large selection of printer drivers. Requires shortwave receiver and cassette cable (described in documentation)

HRSDOS ---------- **$25.00**
Move programs and data between DECB and OS-9 disks. Supports RGB-DOS for split DECB/OS-9 hard drives. No modifications to system modules (CC3Disk or HDisk) required.

DECB SmartWatch Drivers ---------- **$20.00**
Access your SmartWatch from DECB! New function added to access date/time from BASIC (DATE$). Only $15.00 with any other purchase!

RGBOOST ---------- **$15.00**
Make the most of your HD6309 under DECB! Uses new 6309 functions for a small gain in speed. Compatible with all programs tested to date! Only $10.00 with any other purchase!

Robert Gault
832 N. Renaud
Grosse Pointe Woods, MI 48236
313-881-0335
Add $4 shipping & handling per order

Summertime is "off" season for a lot of CoCoists. If you are one of those, look forward to new releases and upgrades this fall. If you use your CoCo all year 'round, the following titles are currently available:

CoCoTop version 1.0	$24.95
CoCoTop version 1.1	$19.95
CoCoTop 1.1 + Tools 3	$34.95
OScopy/RScopy	$10.00
TOOLS 3 version 1.1	$29.95
Quickletter version 2.0	$19.95
Accounting level 2	$34.95
Investing level 2	$24.95
Level II graphics 1.2	$34.95

upgrades only $5.00 (return original disk)
Shipping+handling: US/Canada $3.00 all others $5. Prices in US dollars Send cheque or money order NO COD'S. Call or write for Canadian dollar prices. *Mention the name of this magazine in your order and you will receive a free bonus disk!*

C. Dekker ... *User-friendly Level II Programs!*
RR #4 Centreville, NB
E0J 1H0, CANADA
Phone 506-276-4841

Peripheral Technology Specials

486SLC/33MHz Motherboard w/CPU	$139.00
486SLC/50MHz IBM, ISA,CPU, 0K	$199.00
486SLC/66MHz IBM, VESA, CPU, Math	$299.00
IBM boards - Made in USA - 3YR warranty	
1MB SIMM 70ns DRAM	$47.00
356MB Samsung IDE Drive	$229.00
420MB Connor IDE Drive	$275.00
546MB Maxtor IDE Drive	$379.00
IDE/Floppy/Serial/Parallel	$24.95
1.44MB TEAC Floppy	$49.95
Mini Tower, 200W, LED readout	$79.00
Panasonic Dual Speed CD ROM	$169.00
VGA Card ET4000-1MB, 1280x1024	$99.00
VGA Monitor WEN .28mm 1024x768	$249.00

UPS Ground $7.00 on most items except Tower & monitor ($12.00 UPS Ground).

1250 E. Piedmont Rd. 404/973-2156
Marietta, GA 30062 FAX: 404/973-2170

The OS-9 User's Group, Inc.

Working to support OS-9 Users

Membership includes the Users Group newsletter, MOTD, with regular columns from the President, News and Rumors, and "Straight from the Horse's Mouth", about the use of OS-9 in Industrial, Scientific and Educational institutions.

Annual Membership Dues:

United States and Canada	Other Countries
25.00 US	30.00 US

The OS-9 Users Group, Inc.
6158 W. 63d St. Suite 109
Chicago, IL 60638
USA

Last Issue's "microdisk"

Diretcory of **Volume 2 Number 1** "microdisk". Single issues are available for $6 each (US). Please specify volume and number when ordering. See inside front cover for subscription rates and pricing for other countries.

Disk BASIC Side:
- B09DECB.BAS
- B09QB.BAS
- PICPCX.BAS
- RECVPCX.BAS

OS-9/OSK Side:
- headlines.c
- news_reader.c
- playsndm.c
- playsnds.c
- Gem.ar
- Lander.ar
- Snake.ar

Gem.ar, Lander.ar, and Snake.ar are share-ware programs contributed by subscriber Jean-Pierre Fisette of Canada. If you have written any share-ware programs and would like to have them distributed via microdisk, please send them in! We have 35 and 40 track (double sided) drives for DECB, those plus an 80 track 3.5" drive for OS-9.
If you run across public domain or share-ware programs that you would like to share, feel free to send them in also.

ADVERTISER'S INDEX: